HEVs
haute école valaisanne
hochschule wallis
sciences de l'ingénieur

Filière Systèmes industriels

Orientation Infotronics

Diplôme 2006

Pierre-André Tapparel

CDMS pour cubesat

HEVs
Route du Rawyl 47
1950 Sion 2
haute école valaisanne
hochschule wallis

Professeur Christophe Bianchi

Expert Olivier Moulin

Sion, le 24 novembre 2006

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences
Western Switzerland

route du Rawyl 47 – case postale 2134 – ch-1950 sion 2
tél. +41 (0)27 606 85 11 – fax +41 (0)27 606 85 15
e-mail: info.sion@hevs.ch – internet: www.hevs.ch

Tapparel, Pierre-André 2006

CDMS pour Swisscube

CDMS für Swisscube

Objectif

Etudier et réaliser le cœur, à savoir le CDMS (Control & Data Management System), d'un picosatellite de forme cubique de 10 cm de côté, ne pesant qu'un kilo. Ce picosatellite développé conjointement avec l'EPFL et la HES-SO sera lancé en 2008.

Résultats

L'architecture générale du Swisscube a été écrite en collaboration des autres sous-systèmes. Le design détaillé du CDMS, architecture, choix des composants, schématiques, layout du PCB a été réalisé. Un prototype à base du processeur AT91M55800A a été développé mais n'a cependant pas pu être testé.
Un environnement de développement basé sur des outils GNU a été déployé afin de tester ce prototype.

Mots-clés

CDMS, OBC, Swisscube, cubesat, satellite, ATMEL AT91M55800A, processeur ARM, GNU, Eclipse

Ziel

Untersuchen und verwirklichen das Herz, nämlich das CDMS (Control & Data Managment System), von einem kubischen Picosatellit mit 10 cm Seiten, der nur ein Kilo wiegt. Dieser mit der EPFL und den HES-SO entwickelte Satellit im Jahr 2008 abgeschossen.

Resultate

Die allgemeine Architektur Swisscube wurde in Zusammenarbeit mit den anderen Sub-Systeme geschrieben. Das detaillierte Design des CDMS, die Architektur, die Wahl der Bestandteile, die Schematik und der layout der PCB wurde verwirklicht. Ein Prototyp mit dem AT91M55800A-Prozessor wurde entwickelt aber konnte jedoch nicht getestet werden.
Eine auf GNU-Werkzeugen basierte Entwicklungsumwelt ist entfaltet worden, um diesen Prototyp zu testen.

Schlüsselwörter

CDMS, OBC, Swisscube, cubesat, satellite, ATMEL AT91M55800A, ARM-Prozessor, GNU, Eclipse

Phase B

CDMS

Prepared by:

Tapparel Pierre-André

Checked by:

Approved by:

•
HEVs
Sion
Switzerland
•
11/23/2006
•

HEVs

RECORD OF REVISIONS

ISS/REV	Date	Modifications	Created/modified by
1/0	23.11.2006	First Edition	Tapparel Pierre-André

1 FOREWORD

As the project is divided in different groups from different region, a specific infrastructure was set up in order to exchange information.

The document S3-A-INFR-1-2-Infrastructure.doc presents the organization of concurrent design session and the Swisscube data exchange portal which can be accessed at the following address: http://swiss-cubesat.sharepointspace.com/.

The following figure presents the organization of the different subsystems.

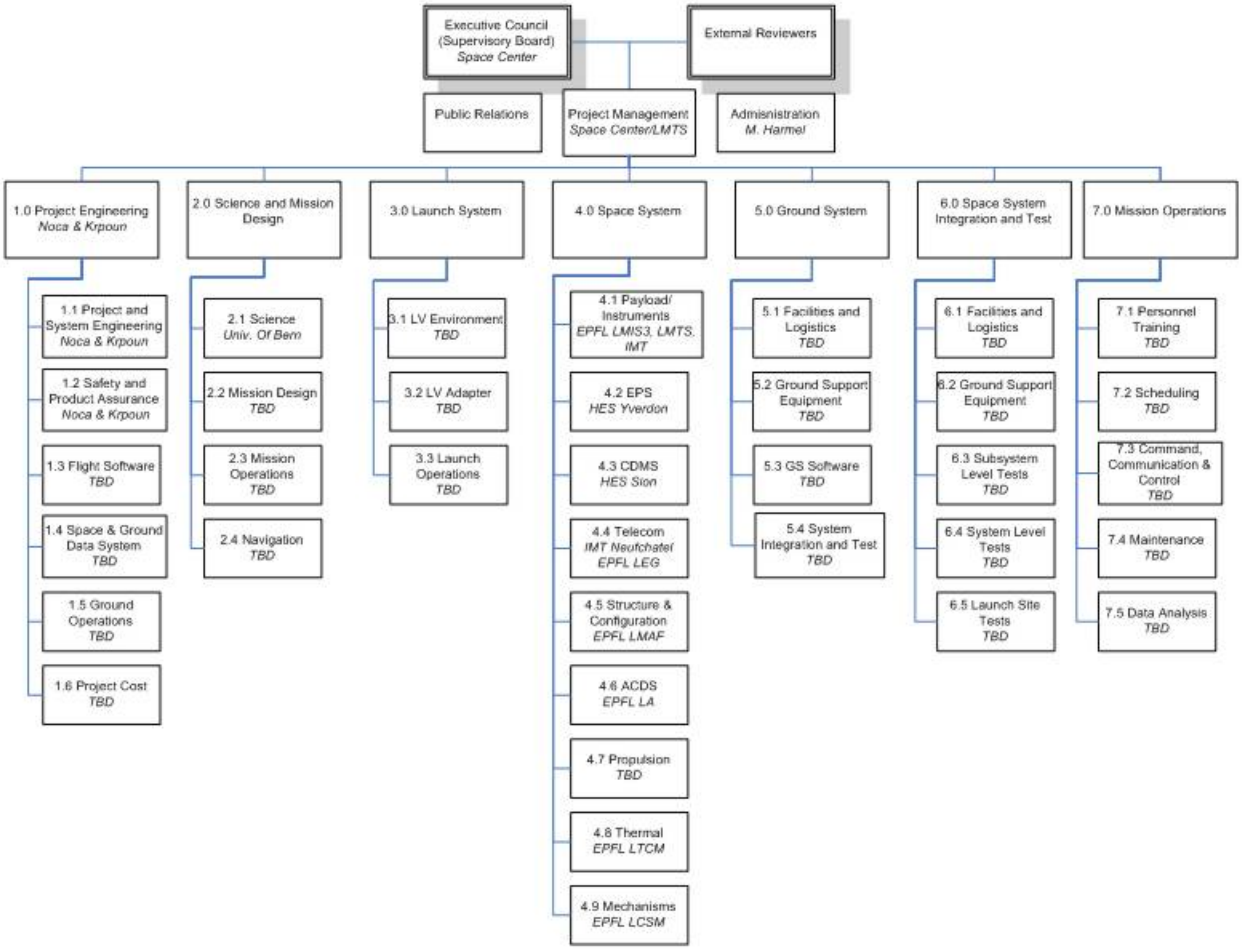


Figure 1 : Swisscube organization

2 INTRODUCTION

This paper is about the construction of a student satellite, named Swisscube. This satellite is categorized as a picosatellite, which refers to the small size of the satellite. Commercial satellites are although varying in size, usually quite big and weighing in the range of 1000 kg's. Our satellite is restricted to 1 kg and a size of 10x10x10 cm - and is therefore referred to as a cubesat. Because of its small size and low weight, many cubesats can be placed in the same launch as secondary payloads. Because of our secondary status, we have no influence on the orbit our satellite will have, we have to do with whichever orbit the companies of the primary payloads select.

Because of the relatively humble conditions mentioned above, the price of getting the satellite in space is very low compared to the prices of commercial satellite launches.

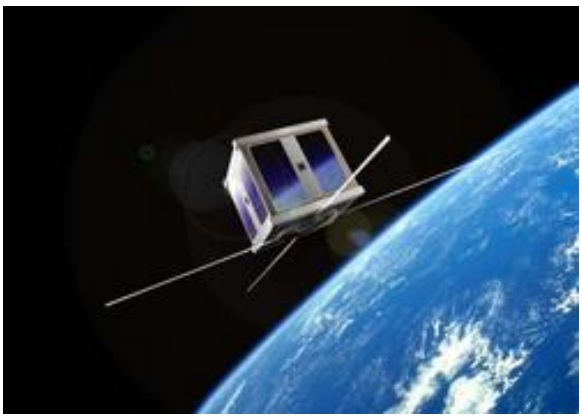


Figure 2 : Picture of a cubesat

As building a satellite is a very big project, the various tasks of designing the different parts have been assigned to different groups.

Our group consists of two people working in the University of Applied Science (HEVs) based in Sion, Switzerland. The first one is a student working on the project during his project of diploma, and a supervisor, who is professor at the HEVs. Our tasks consist in designing the control and data-management system, and provide hardware expertise to other groups.

At it is imperative that the different parts function together, communication between the groups is of outmost importance. Budgets have to be made, interfaces have to be defined and the knowledge contained in the groups has to be shared in the best possible way in order to hope for a successful mission. The different groups meet once a week to discuss various issues, progresses and setbacks.

3 REFERENCES

3.1 Normative references

[N1] Preliminary Functional Specification E3-AB-SEIC-1-0-Prel_Functional_Spec

[N2] Design Activities Overview & Required Infrastructure: S3-A-INFR-1-2-Infrastructure.doc

[N3] "The I2C bus specification, version 2.1, January 2000: Philips doc. N°9398 393 40011

[N4] CubeSat Design Specification - Revision 9 from Calpoly

[N5] ATMEL_PLL_LFT_Filter_CALCULATOR_AT91.xls

3.2 Informative references

[R1] CDMSPhaseBreview_v1.0.doc

[R2] "The CAN bus - An introduction" course, Dominique Gabioud, HEVs

[R3] S3-A-EPS-1-0-Latch-up_French

[R4] ECSS-Q-70-11A

[R5] MIL-HDBK-217F

4 TERMS, DEFINITIONS AND ABBREVIATED TERMS

4.1 Abbreviated terms

ADC	:	Analog to Digital Converter
ADCS	:	Attitude Determination and Control System
AIC	:	Advanced Interrupt Controller
API	:	Applications Programming Interface
APMC	:	Advanced Power Management Controller
BCD	:	Binary Coded Decimal
Bps	:	Bits per second
BYTE	:	8-bit
CAN	:	Controller Area Network
CDMS	:	Control and Data-Management System
CMOS	:	Complementary Metal-Oxide Semiconductor
COFF	:	Common Object File Format
CoG	:	Centre of Gravity
COM	:	COMmunication Subsystem
COTS	:	Commercial Off-The-Shelf
CRC	:	Cyclic Redundancy Code
DAC	:	Digital to Analog Converter
DOD	:	Department Of Defense
EBI	:	External Bus Interface
EDAC	:	Error Detection And Correction
ELF	:	Executable and Linking Format
EMC	:	ElectroMagnetic Compatibility
EMI	:	ElectroMagnetic Interference
EOF	:	End Of Frame
EPS	:	Electrical Power Subsystem
ESD	:	ElectroStatic Discharge
GNU	:	GNU's Not Unix
HAL	:	Hardware Abstraction Layer
HALFWORD	:	16-bit
I^2C	:	Inter-Integrated Circuit

ICE	:	In Circuit Emulation
ISI	:	Inter-Symbol Interference
JTAG	:	Join Test Action Group
MCU	:	MicroController Unit
MSB	:	Most Significant Bit
NRZ	:	Non-Return to Zero
OBC	:	On Board Computer
OSI	:	Open System Interconnection
PCB	:	Printed Circuit Board
PDC	:	Peripheral Data Controller
PL	:	Payload
PLL	:	Phase Locked Loop
RAM	:	Random Access Memory
ROM	:	Read Only Memory
RTC	:	Real Time Clock
SAR	:	Successive Approximation Register
SCL	:	Serial Clock Line
SDL	:	Serial Data Line
SMBus	:	System Management Bus
SNR	:	Signal to Noise Ratio
SOF	:	Start Of Frame
SPI	:	Serial Peripheral Interface
TID	:	Total Ionizing Dose
TQFP	:	Thin Quad FlatPack
TTL	:	Transistor-Transistor Logic
USART	:	Universal Synchronous/Asynchronous Receiver-Transmitter
WORD	:	32-bit

5 DESIGN DRIVERS FOR THE CDMS

5.1 Swisscube topology

The following figure presents the last iteration done for the Swisscube topology. It has been decided to centralize all the functions that require a lot of processing capacities on one subsystem, the CDMS.

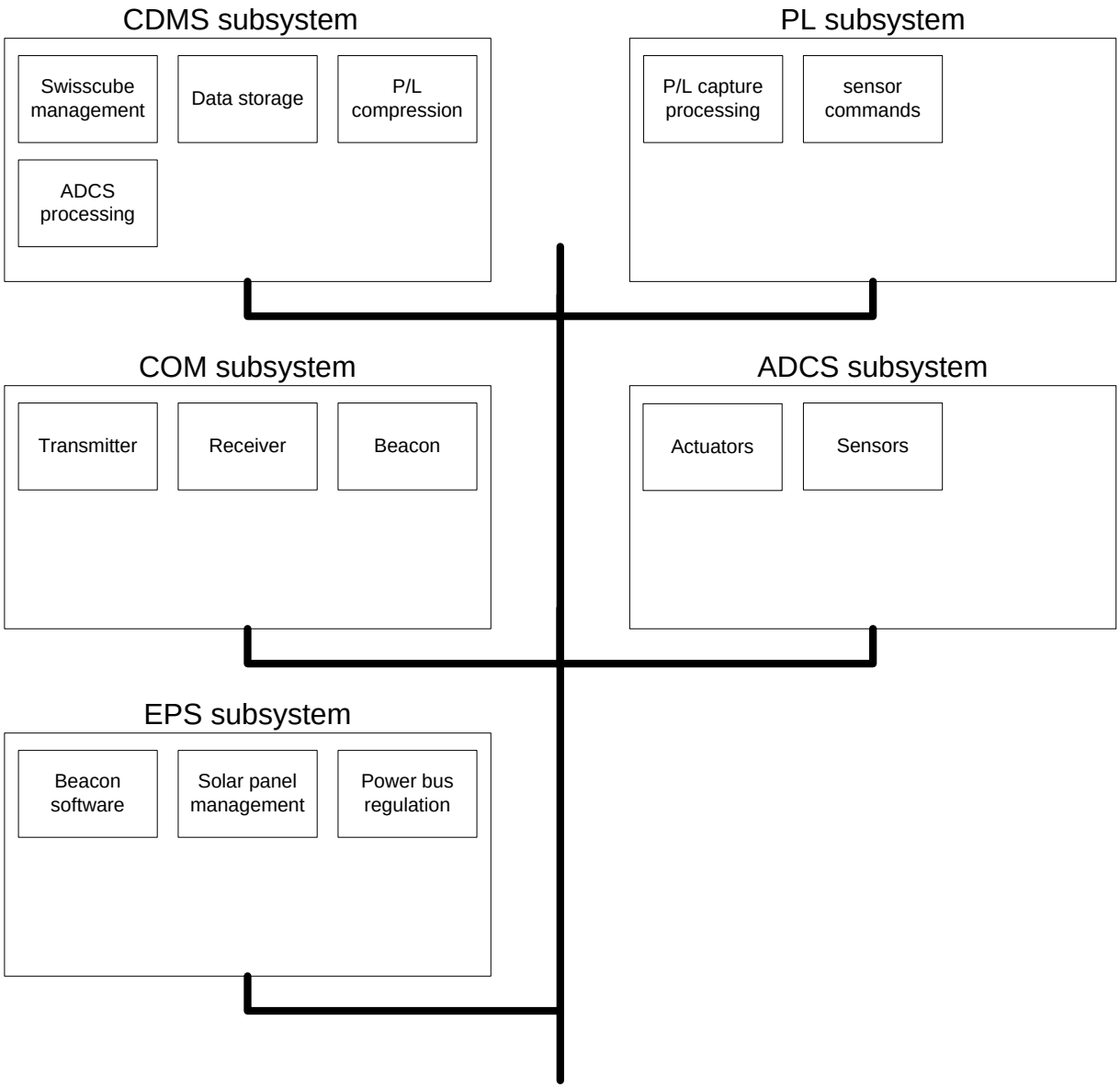


Figure 3 : Swisscube topology

5.2 Sub-system requirements

In order to have a complete and fully functional satellite, it is imperative that all groups comply with a set of specifications, so that the resource in terms of space and power are used in the best manner without exceeding the budgets. The different groups, who individually have made preliminary budgets for their own parts, have formed these specifications and budgets. These budgets have been summed up in system-engineering perspective, mainly by the power and the mechanics group, in order to have an idea of whether or not they are realistic. The final measures are still to be determined, but below the present guidelines are listed.

All the following subsystem must provides a number of A/D converters, digital I/O and sufficient memory to store software.

5.2.1 Electrical power subsystem (EPS)

The microcontrollers take measures of solar array current; calculate the slope between to measures and chose the best duty cycle to command the step-up converter in order to have the maximum power. The step-up converter frequency will be about 100[kHz] .This microcontroller must execute the software of beacon subsystem too.

5.2.2 Communications subsystem (COM)

In order to send and receive data from the ground station, the satellite must have a communication subsystem. The COM subsystem will be based on the same microcontroller then EPS. This microcontroller should provide an interface to be connected on the main bus, in the way to communicate with other subsystem.

5.2.3 Control and data-management system (CDMS)

The CDMS must provide a 16[Mbit] long memory in order to store the science images took by the payload until the data can be transferred to the ground station. Moreover, 16[Mbit] must be available in order to save command and other information, and to offer the possibility to upload new application software. This subsystem should provide the ability to made housekeeping, scheduling and interprets different command send from the ground station. The CDMS will provide sufficient processing capacities in order to make the compression of images captured by the payload, to calculate the attitude control for the ADCS, and to run the embedded operating system.

5.2.4 Attitude determination and control system (ADCS)

The microcontroller included on the ADCS board will manage the sensors and actuators in the way to control the attitude of the satellite, all the processing will be done on the CDMS microcontroller.

5.2.5 Payload (PL)

The payload will take picture of nightglow effects; the interface between the image sensor and the microcontroller is not yet defined. The assumption was made that the PL board will integrate a microcontroller which will manage this sensor, but it is also possible to implement this function directly on the CDMS microcontroller. The images captured must be compressed, and this compression will be made on the CDMS.

5.2.6 Summary

With those requirements, we see that two different microcontroller must be used, one powerful and one other which will manage simpler functionality. The small microcontroller will be used by different subsystem which requirements should differ. In order to simplify the architecture and spatial test, only one type of microcontroller for the four different subsystems.

5.3 Operating system requirements

As it will be interesting to have an operating system on the CDMS, some OS for embedded system has been compared in order to have an idea of the memory and processing requirements. This section will not present which OS is the best one for our application, this selection will be made by the Flight Software group. Four different OS has been compared; eCos, µClinux, RTEMS and VxWorks.

5.3.1 eCos

Characteristics	Supported processors	Required memory
- Real time / Multithread - System Configuration (flexible, little place in memory) - Free software (GNU license) - HAL (portability, simplicity of programming, more independent software)	- ARM - PowerPC - Hitachi H8300 - Intel x86, MIPS - Matsushita AM3x - Motorola 68k - SuperH - SPARC - NEC V8xx	- Required memory to make the minimal infrastructure running (the core, support of peripherals and the memory): from ten to hundred kilobytes.

5.3.2 µClinux

Characteristics	Supported processors	Required memory
- Free - API compatible with Linux (many software already developed and compatible free codes) - Many documentation (Linux) - GNU tools (compiler and debugger) - Important basic functionalities	- Motorola DragnBall - Motorola ColdFire - ADI Blackfin - ETRAX - ARM7TDMI - Intel i960 - Atari 68k - NEC V850E - H8	- Core size lower than 512[kB] - Core size and Linux commands, lower than 900[kB]

5.3.3 RTEMS

Characteristics	Supported processors	Required memory
 - Hard real time system, multithreads and multiprocessors - Developed by the army, widely currently used (communications, medical field, space and aviation, robotics…) - Use of RTEMS encouraged by ESA - RTEMS is free and subject to GNU license. - Environment usage (GNU tools) available with RTEMS.	- AMD A29K - Motorola MC68xxx - Motorola Coldfire - ARM - MIPS R3000 and R4000 - Open Cores OR32 - Texas instrument C3x / C4x - Renesas SHx - PowerPC	- Core size lower than 70[kB] with the libc posix

5.3.4 VxWorks

Characteristics	Supported processors	Required memory
VxWorks® - World leader of the optimization of software for peripherals - Proprietary system and paying -Multithread and real time (advanced communication between thread) - flexible - more than 800API available	- Freescale/IBM PowerPC - Freescale ColdFire - Freescale ColdFire - Intel StrongArm and XScale - MIPS - ARM - Hitachi SuperH	- The system is, with the same manner as eCos, flexible; therefore the report needs vary according to desired applications. - However, the memory necessary to make the minimal infrastructure running is a few ten to hundred kilo bytes

Excepting the µClinux, we can see, with this comparison, that a memory of appreciatively 200[kB] will be sufficient to save the operating system. 512[kB] will be installed on the CDMS in order to save the OS and the application software developed.

5.4 Cubesat OBC (On Board Computer) Overview

In the way to have an idea about was has been already done by the cubesat community, this section presents some information about other cubesat's on board computer.

5.4.1 AAUSat-I

Specification	Data
Weight	
Operating Voltage	5[V]
Processor	Siemens C161-RI
Clock Frequency	10[MHz]
Memory	RAM: 512[kB]
	ROM: 512[kB]
	Flash: 256[kB]
Bus	DMA and I2C
Operating system	RTX166
Programming language	C

5.4.2 AAUSat-II

Specification	Data
Weight	
Operating Voltage	3,3[V] (<300[mW] @ 40[MHz]; <80[mW] @ 8[MHz])
Processor	Atmel AT91SAM7A1
Clock Frequency	8 and 40[MHz]
Memory	RAM: 2[MB] SRAM
	Flash: 2*4[MB]
Bus	CAN
Operating system	
Programming language	

5.4.3 CanX-I

Specification	Data
Weight	
Operating Voltage	3,3[V] (0,4[W])
Processor	Atmel ARM7
Clock Frequency	40[MHz]
Memory	RAM: 512[kB] SRAM
	ROM: 128[kB]
	Flash: 32[MB]
Bus	One-wire, serial RS232
Operating system	No operating system
Programming language	C

5.4.4 CP1 and CP2

Specification	Data
Weight	
Operating Voltage	3[V]
Processor	PIC18LF6720
Clock Frequency	4[MHz]
Memory	ROM: at least 1[kB]
	RAM: 4[kB]
	Flash: 128[kB]
Bus	I2C
Operating system	
Programming language	

5.4.5 CubeSat Kit

Specification	Data
Weight	
Operating Voltage	3,3 and 5[V] (<100[mW])
Processor	Texas Instrument MSP430
Clock Frequency	
Memory	SD/MMC external memory
Bus	I2C, SPI, USART

Operating system	Salvo Pro RTOS
Programming language	MSP430 C Compiler

5.4.6 CubeSat XI-IV

Specification	Data
Weight	
Operating Voltage	2.0 and 5.5[V]
Processor	PIC16F877
Clock Frequency	4[MHz]
Memory	RAM: 368 bytes
	ROM: 32[kB]
Bus	I2C
Operating system	
Programming language	

5.4.7 CUTE-I

Specification	Data
Weight	
Operating Voltage	
Processor	H8/300
Clock Frequency	
Memory	RAM: 1[kB] internal RAM, 512[kB] external SRAM
	ROM: 256[kB] EEPROM
	Flash: 4[MB] AT45DB321B
Bus	
Operating system	
Programming language	

5.4.8 DTUSat

Specification	Data
Weight	63g]
Operating Voltage	3,3[V] (3[mW/MHz])
Processor	Atmel AT91M40800
Clock Frequency	16[MHz]
Memory	ROM: 16[kB]

	RAM: 1[MB]
	Flash: 2[MB]
Bus	SPI
Operating system	
Programming language	

5.4.9 HAUSAT-1

Specification	Data
Weight	<45[g]
Operating Voltage	3,3 and 5[V] (60[mW])
Processor	AT91LS8535
Clock Frequency	4[MHz]
Memory	RAM: 512 bytes internal SRAM
	ROM: 512 bytes EEPROM
	Flash: 4[MB] AT45DB321B
Bus	SPI
Operating system	
Programming language	IAR C Compiler

5.4.10 KUTEsat

Specification	Data
Weight	55[g]
Operating Voltage	3,3[V]
Processor	PIC18F4220
Clock Frequency	
Memory	SDRAM: 8[MB], Flash: 4[MB]
Bus	
Operating system	
Programming language	

5.4.11 Mea Huaka'i

Specification	Data
Weight	
Operating Voltage	5 ± 0,25[V]
	RCM2000: High 0,65[W]; Med 0,3[W]; Sleep 1,4[mW]

	EEPROM: Write 165[mW], Read 2,2 [mW], Standby 0,55[µW]
Processor	Z-World RabbitCore RCM2000 and 2300 (x2 for redundancy)
Clock Frequency	1,8 – 30 [MHz]; 32[kHz] in sleep mode
Memory	
Bus	I2C
Operating system	
Programming language	Dynamic C

5.4.12 MEROPE

Specification	Data
Weight	
Operating Voltage	
Processor	Motorola MC68HC812A4
Clock Frequency	
Memory	128[kB] Supersync FIFO IDT72291
Bus	
Operating system	
Programming language	

5.4.13 NCUBE

Specification	Data
Weight	
Operating Voltage	3,6[V]
Processor	Atmel AVR ATmega32L and PIC microcontrollers
Clock Frequency	
Memory	ROM: 4[kB]
Bus	I2C
Operating system	
Programming language	

5.4.14 Sacred and Rincon

Specification	Data
Weight	
Operating Voltage	
Processor	PIC16C77

Clock Frequency	4[MHz]
Memory	RAM: 368 byte on chip; 64[kB] FRAM
Bus	I2C
Operating system	
Programming language	C

A wide range, in term of performance, of OBC has been developed. It goes from a small 8-bit microcontroller with a few kilobytes memory to a 32-bit microcontroller with some megabytes memory. The architecture of the on-board computer greatly depends on the functionality required by the others subsystems. It is however possible to see that the PIC family microcontroller and the I^2C bus is widely used. For the future development of the CDMS, this information will be useful in order to see, as far as possible, if some components could be used.

5.5 Environment

During the preparation for launch and then during the flight, the spacecraft is exposed to a variety of mechanical, thermal, and electromagnetic environments. This chapter provides a description of the environment that the spacecraft is intended to withstand. For more details please refer to [R1].

5.5.1 Radiation effects on electronic

Radiation in space is generated by particles emitted from a variety of sources both within and beyond our solar system. The nature of the radiation differs in place and time. In some places our satellite might cross the so-called 'solar wind', which is a burst of particles (mostly protons and electrons) emitted from the sun, and here radiation will be substantial. In other places radiation will be very light. The single particles also vary in the amount of energy they possess. High-energy particles are more dangerous to our circuitry than low-energy particles (often referred to as background radiation), as they penetrate deeper into the components and cause greater damage in case of collision. Much research is carried out in making maps of intensities and different kinds of radiation in space. Such a map is show in the figure 1. The figure shows that the radiation is not only depending on the sun but also very much on the magnetic field of earth (magnetosphere).The orbit of our project is so low that one will not touch the Van-Allen belts (turquoise zone).As further details of this area are quite complex and out of the scope of this project, we will not discuss it further.

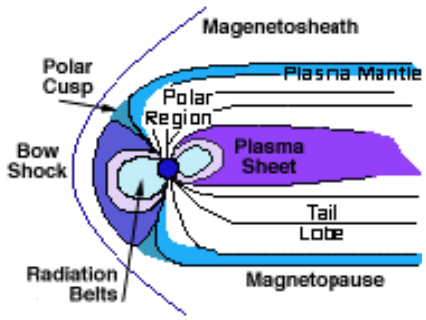


Figure 4 : Radiation near the earth

5.5.1.1 Single event effects

A single-event effect results from, as the term suggests, a single, energetic particle. If such a particle hits one of our components, we might encounter three different errors:

- Bit-flips (soft error)

This phenomenon causes the value of a specific bit to change. This will lead to software malfunction. In order to avoid this malfunction we plan to implement error detection and correction circuitry (EDAC) between the processor and the memory chips. This will be able to correct most errors. In case of substantial damage to the software, it might be necessary to reboot or even upload new software from earth. The ROM has to be radiation-hard so that it does not suffer from these bit-flips.

- Latch-up (hard error)

The Figure 2 shows a part of a CMOS component being hit by a high-energy particle. This particle can either short-circuit the source or drain to ground, which is located in the bottom of the figure. The short circuit arises because the particle creates a conductive 'tunnel'. A short circuit will draw a lot of current and the component is very likely to take permanent damage. The issue is to include a latch-up protection circuitry near the sensible component to turn off the power immediately.

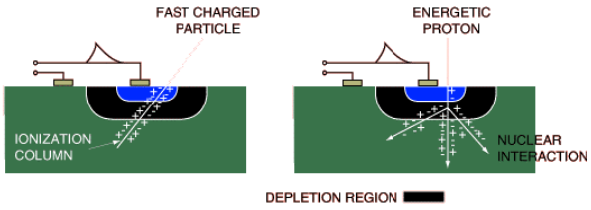


Figure 5 : Particle colliding with a CMOS Component

- Burn-out (hard error)

In case of the power not being turned off at a latch-up, a burn-out can occur. A burn-out can sometimes be seen by the human eye, as a burn-out means that the chips is simply melted in the area of the latch-up. After this, the chip is useless.

5.5.1.2 Total ionizing dose (TID)

The total ionizing dose, mostly due to electrons and protons, can result in device failure. TID is measured in terms of the absorbed dose. The TID is calculated from the trapped protons and electrons, secondary Bremsstrahlung photons, and solar flare protons. As TID increase, material degradation increase. Long-term exposure can cause device threshold shifts, increased device leakage and power consumption, timing changes, decreased functionality, etc.

In order to have an idea of how our components will react to these long-term effects, a sample of component should be tested. The amount of radiations is measured in kRad. 5 to 10 kRad is the average amount of radiation we can expect in one year on the orbit used by our cubesat. The duration of the spatial mission is set as three month. The components should so be able to work wit a TID of 3[kRad]. This factor should not play a major role for our short mission; nevertheless the component must be tested.

The only thing that can be done to limit these long-term effects is to place shielding material around the components. This could for example be aluminium, which is both light and well shielding. This kind of shielding will not stop any of the high-energy radiation that causes latch-ups and bit-flips.

5.5.2 Thermal

The environment that the spacecraft experiences both during its preparation and once it is encapsulated under the fairing is controlled in terms of temperature, relative humidity, cleanliness, and contamination.

For the Vega, Soyuz and Ariane 5 LVs, the typical thermal environment within the air-conditioned GSC facilities is kept around 23°C ± 2°C for temperature and 55% ± 5% for relative humidity.

Before fairing jettisoning, the thermal flux density radiated by the fairing does not exceed at any point 1000 W/m^2 for Vega and Ariane 5, and 800 W/m^2 for Soyuz. For the complete estimation of the thermal environment under the fairing the spacecraft dissipated power shall be taken into account.

After fairing jettisoning, the nominal time for jettisoning the fairing is determined in order not to exceed the aerothermal flux of 1135 W/ m^2 for Vega, Soyuz and Ariane 5. Typically the aerothermal flux varies from 1135 W/ m^2 to less than 200 W/ m^2 within 20 seconds after the fairing jettisoning, as presented in the following figure.

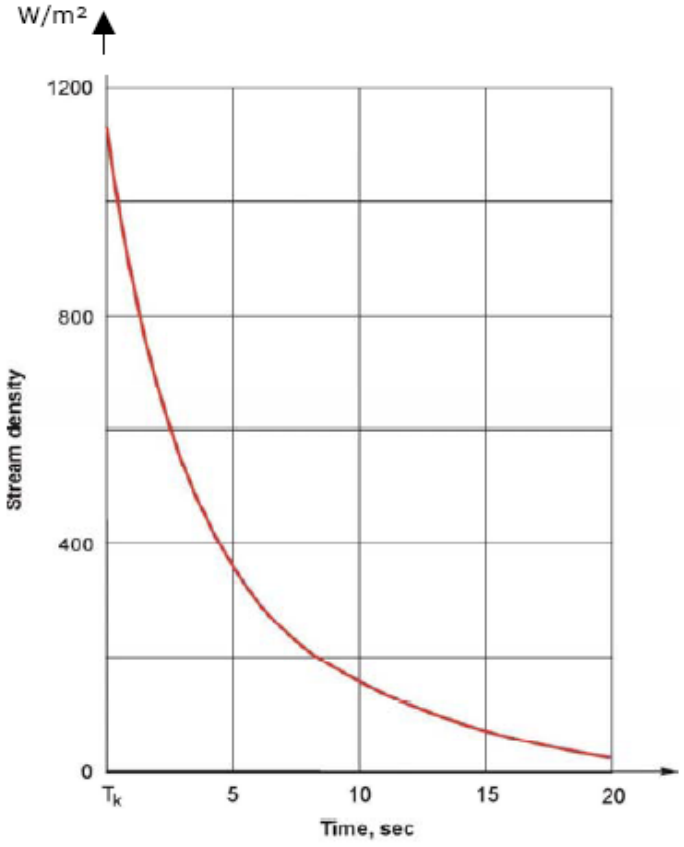


Figure 6 : Typical Aerothermal Flux Decay after fairing jettisoning.

5.5.3 Mechanical

The dimension of the cubesat must follow the specification given by the reference [N4]. Moreover, during flight, the spacecraft is subjected to static and dynamic loads. Such excitations may be of aerodynamic origin (e.g., wind, gusts, or buffeting at transonic velocity) or due to the propulsion systems (e.g., longitudinal acceleration, thrust build-up or tail-off transients, or structure-propulsion coupling, etc.). Those different aspect are presented in reference [R1]

6 SWISSCUBE BUS TOPOLOGY

As we decided to build a distributed architecture, distributed architecture means that all system have their own microcontroller and do their own job, all subsystem must be able to communicate with each-other. In order to transmit data or other information, a bus must be implemented between the subsystems. The following topology has been decided.

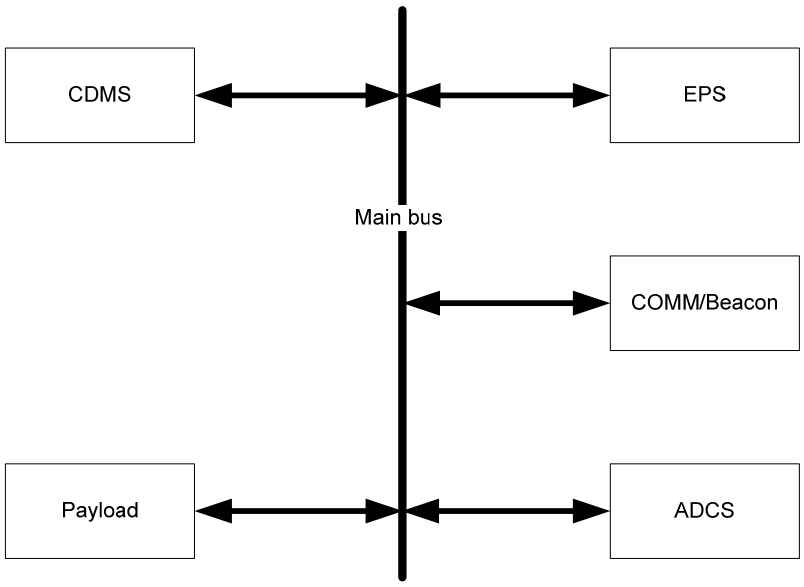


Figure 7 : Bus topology overview

6.1 Main bus

The main bus will provide the communication between CDMS, EPS, COMM, Payload and ADCS.

In order to simplify the design and to keep the number of different interfaces to learn low, we decided to have only one bus which all subsystems will be connected on.

As command and important measure will be transferred on the main bus, the assumptions were made that the main bus will be reliable and provide enough data rate. Moreover as five subsystems will be connected on the main bus, the protocol will provide multipoint topology.

The communications between those subsystems consists in sending command from the CDMS to other subsystems and receive measures from other subsystems to the CDMS in order to save them. Moreover the Payload will send captured images to the CDMS. The size of one image transferred is

about 150[kbit], and one image will be transferred every 30[sec] during the science mode of the satellite. During the mode while the satellite is sending data to the Ground station, the data rate is limited by the capacity of the COM subsystem, which is about 1.2[kbps].

The communication protocol should be reliable enough, in the way to keep the satellite working properly if some command will be corrupted or loosed during communication between the subsystems.

6.2 Type of bus

In computer architecture, a bus is a subsystem that transfers data or power between computer components inside a computer or between computers.

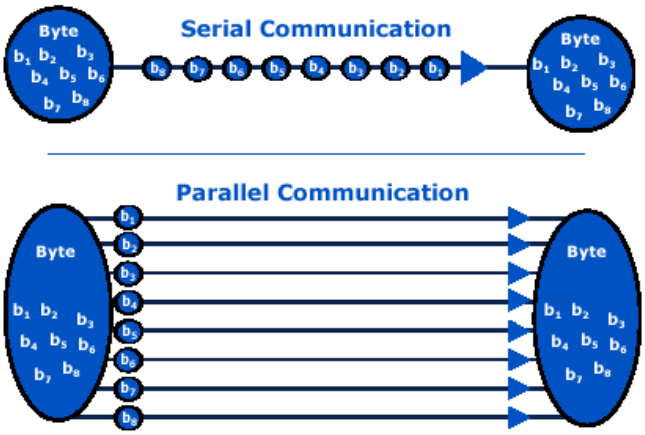


Figure 8 : Serial vs. parallel communication

In a digital communications system, there are two methods for data transfer: parallel and serial. Parallel connections have multiple wires running parallel to each other (hence the name), and can transmit data on all the wires simultaneously. Serial, on the other hand, uses a single wire to transmit the data bits one at a time.

It is a natural question to ask which one of the two transmission methods is better. At first glance, it would seem that parallel ports should be able to send data much faster then serial ports. Let's say we have a parallel connection with 8 data wires, and a serial connection with a single data wire. Simple arithmetic seems to show that the parallel system can transmit 8 times as fast as the serial system.

However, parallel ports suffer extremely from inter-symbol interference (ISI) and noise, and therefore the data can be corrupted over long distances. Also, because the wires in a parallel system have small amounts of capacitance and mutual inductance, the bandwidth of parallel wires is much lowers than the bandwidth of serial wires. An increased bandwidth leads to a better bit rate. We also know that less noise in the channel means we can successfully transmit data reliably with a lower signal to noise ratio (SNR). In addition, because of the increased potential for noise and interference, parallel wires need to be far shorter than serial wires. Moreover, as the bus will connect the different subsystems, it will be preferable to use less wire than possible.

In conclusion, for our use, it is preferable to use a serial link between the subsystems.

6.3 Bus candidates

As shown before, the serial link will provides the best choice for our needs. All the following bus use serial link. The descriptions following are only summaries which present the main features of the bus. The OSI layer model is presented on Appendix A.

- UART (universal Asynchronous Receiver Transmitter)

The UART is a piece of computer hardware that translates between parallel bits of data and serial bits.

The four following bus are based on the UART component, but used with different drivers.

- driver RS232

  The RS232 specification defines Mechanical, Electrical, and Functional characteristics. RS232 is an Unbalanced (Single Ended), unidirectional (point-to-point) interface. Signal is referenced to ground. RS232 drivers feature a controlled slew rate. Normal output levels are ± 5 [V]. RS232 use asynchronous framing with a known data width of 8bits, and NRZ (non-return to zero) encoding.

- driver RS422

  RS422 define a balanced (differential) interface; specifying a single, unidirectional driver with multiple receivers (up to 32). RS422 will support Point-to-Point, Multi-Drop circuits, but not Multi-point.

- driver RS485

  RS485 define a balanced (differential) interface; defines the Physical layer (OSI Layer 1), signaling protocol is not defined. RS485 specifies bidirectional, half-duplex data transmission. Up to 32 transmitters and 32 receivers may be interconnected in any combination, including one driver and multiple receivers (multi-drop), or one receiver and multiple drivers. RS485 is the Multi-Point version of RS422.

- driver LVDS EIA-644

  LVDS as Low Voltage Differential Signaling defines the electrical layer only. The EIA-644 provides a point to point topology with differential interface.

- I^2C

The I^2C bus uses a bi-directional Serial Clock Line (SCL) and Serial Data Lines (SDL) and due to its two-wire nature can only communicate half-duplex. The interface uses 8 bit long bytes, most significant bit (MSB) first, with each device having a unique address. Any device may be a transmitter or receiver, and a master or slave. Data and clock are sent from the master.

- SMBus

The SMBus (system management bus) is a two wire interface which is based on the I^2C bus. The two SMBus lines are called SMBCLK and SMBDAT and operate at a frequency of 100[KHz]. Both SMBCLK and SMBDAT are bidirectional, and pulled high via a resistor. The SMBus link may have multiple masters and slaves on the bus, but only one master may be active at any one time. Slaves may receive or transmit data to the master.

- SPI (Serial Peripheral Interface)

The SPI Bus is a four wires serial communications interface used by many microprocessor peripheral chips. The SPI circuit is a synchronous, full duplex serial data link setup as a

Master/Slave interface. The SPI bus specifies two control lines. Chip Select (CS) and Serial Clock (SCLK) and two data lines, Serial Data In (SDI) and Serial Data Out (SDO). One SPI device acts as the SPPI Master by controlling the data flow and asserting device select then receives or transmit data via the two data lines.

- CAN bus

The Controller Area Network (CAN) specification defines the Data Link Layer (OSI Layer 2) and the Physical Layer (OSI Layer 1) too.

The CAN bus is a balanced (differential) two-wire interface. This bus use NRZ encoding to ensure compact messages with a minimum number of transition and high resilience to external disturbance.

- Spacewire (IEEE 1355)

The Spacewire bus provides a bidirectional serial interconnect which builds a scalable parallel system using a pair of unidirectional lines. IEEE 1355 defines the Physical and Data Link Layer. The electrical interface is specified as standard Transistor-Transistor Logic (TTL), using either 3.3 or 5[V].

- MIL – STD – 1553

MIL-STD-1553 is a department of defence (DOD) Military (MIL) Standard (STD), which defines both the mechanical, electrical, and functional characteristics. MIL 1553 uses a balanced (differential) interface. The interface is dual redundant with between 2 and 32 interfaces devices on the bus. The multiplex data bus system shall function asynchronously in a command/response mode, and transmission shall occur in a half-duplex manner.

6.4 Analysis criteria

As shown further some criteria has different reference and weighting depending on which use of the bus will be made. The other criteria has the same weight and reference for the two different trade-offs.

- Reliability

The reliability provides a comparison between different buses; this comparison is made upon the immunity to electromagnetic interferences, the possibilities offered to check if errors occur and prioritization it several nodes will transmit at the same time.

It has a weighting of 3.

- Power consumption

In accordance to the embedded system specification, we only have a few watts available, in that way the power consumption is a really important criterion.

Power consumption has the weight of 5.

- Availability on commercial market

As the name says, this criterion compares the provided component supporting the bus, in other words it indicates if the bus is commonly used or not.

The availability has the weight of 4.

- Data rate

The assumption was made that the main bus will provide a high data rate, for this case the reference is set as 1[Mbps] and the weight factor as 3.

- Topology

This criterion indicates if the bus supports multi-point or multi master topology. The main bus requires a multi-point or multi-master topology in that case this factor has the weight of 4.

- Implementation facilities

As implementation facilities, we say if the bus needs specific component like transceiver, buffer, or a specific protocol.

The availability has the weight of 3.

We have also compared the bus on a few others subjects like voltage, development tools and if this bus has already be used in space.

6.5 Bus selection

The following tables present the bus comparison.

Items		Reliability	Power consumption	Availability on commercial market	Data Rate	Vcc 3.3V or 5V	Topology	Already used in space	Implementation facilities	Dvpt tool in school	Results
Reference		CRC/diff	2Tx & 2Rx Active: 25 mW Stdby: 5 mW	Multiple sources	1Mbit	dig.std	Multipoint / Multi MASTER	Industry or cubesat	Without ext. Component / protocol	Facilities	
weighting factor		3	5	4	3	1	4	4	3	3	
Bus	RS232 (UART)	-	++	++	--	++	--	+	++	++	109
	UART (driver RS422)	O	-	O	++	++	-	+	+	O	96
	UART (driver RS485)	O	-	O	++	++	-	O	+	O	92
	UART (driver LVDS EIA-644)	O	--	O	++	++	-	+	+	O	91
	I2C	O	++	++	-	++	++	O	O	++	121
	SMBus	O	++	O	--	++	++	--	O	+	99
	SPI	-	++	+	O	++	++	--	O	+	106
	CAN bus	++	-	+	+	++	++	++	O	++	122
	Spacewire (IEEE1355)	++	--	--	++	++	--	++	--	-	77
	MIL-STD-1553	++	--	--	+	--	+	++	--	--	79

In accordance with the assumptions made, we should see that two different buses appear to be the best choice for the main bus. The IIC and CAN bus which have appreciatively the same results. The main difference between those buses that the CAN bus is based on two differential lines which provides the best reliability but the power consumption of this bus is higher than the IIC bus. Those two different buses will be analyzed more in detail, in order to select the best one for our application.

6.6 CAN bus, how does it work?

The CAN (Controller Area Network) is a serial bus for a network of (micro-) controllers. It was originally developed by R.BOSH for use in automobiles at the end of the 80's. Today it is an ISO (International Standard Organization) standards, cheap, robust, medium speed, and real-time.

As seen before, the CAN bus is a differential two-wire interface which use NRZ encoding and bit-stuffing. Bit-stuffing is to create artificially transition upon long series of 0's or 1's. The transmitter adds a dummy dominant (recessive) after 5 recessive (dominant) symbols. The receiver does not consider the dominant (recessive) following 5 recessive (dominant) symbols.

NRZ Encoding is used in slow speed synchronous and asynchronous transmission interfaces. With NRZ, a logic '1' bit is sent as a high value and a logic '0' bit is sent as a low value. The receiver may lose synchronization when using NRZ to encode a synchronous link which may have long runs of consecutive bits with the same value (no changes in voltage). Other problems with NRZ include; Data sequence containing the same number of 1's and 0's will produce a DC level, and NRZ requires a large bandwidth, from 0[Hz] (for a sequence containing only 1's or only 0's) to half of the data rate (for a sequence of 10101010).

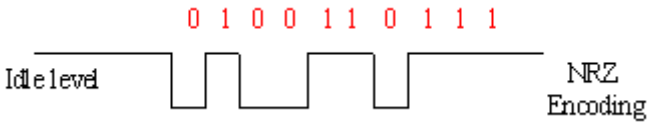


Figure 9 : NRZ Encoding

While CAN uses differential wire, it has a great immunity against electromagnetic interference (EMI), because EMI generates the same changes on the potential of both wires. The receiver evaluates the differential voltage U_{diff} which remains stable.

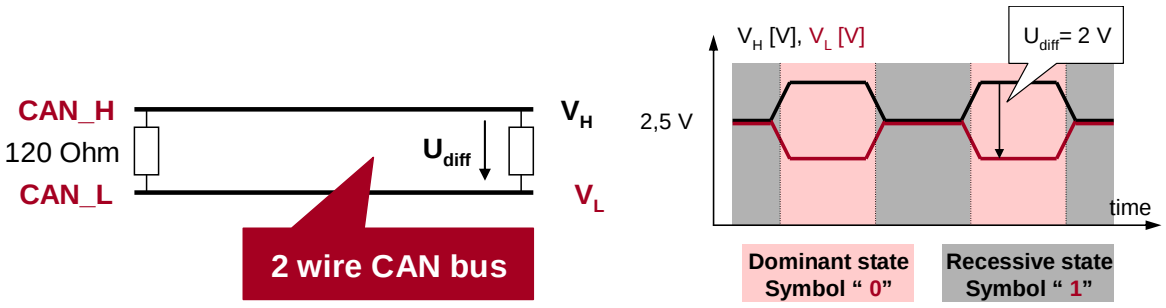


Figure 10 : CAN bus electrical interface circuit

A number of different data rates are defined, with 1[Mbps] (Bits per second) being the top end, and 10[Kbps] the minimum rate. All modules must support 20[Kbps]. Cable length depends on the data rate used. Normally all the device in a system transfer uniform and fixed bit-rates. The maximum line length is 1[Km], 40 meters at 1[Mbps]. Termination resistors, avoiding reflection, are used at each end of the cable.

The CAN Bus interface uses an asynchronous transmission scheme controlled by start and stop bits at the beginning and end of each character. This interface is used, employing serial binary interchange. Information is passed from transmitter to receivers in a data frame.

As from a CAN point of view, all nodes are equivalent; several nodes can transmit at the same time. The message with the lowest Message Identifier gets higher priority. Transmission of the messages with the higher Message Identifier is stopped at some time during the transmission of the Message Identifier. The bus arbitration scheme is called CSMA/CA (Carrier Sense Multiple Access based on Collision Avoidance).

6.6.1 CAN bus message frame

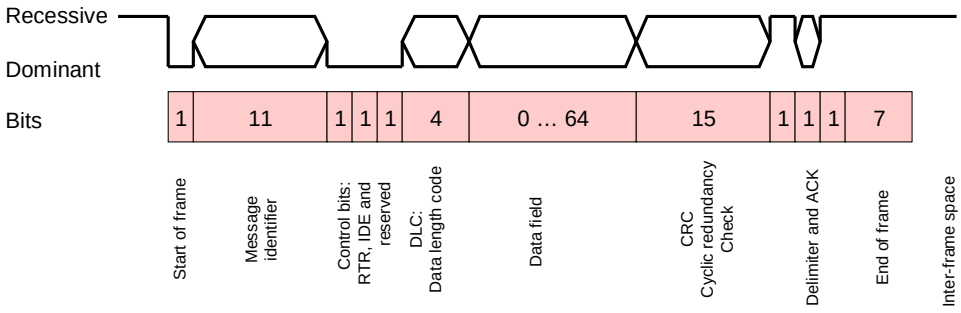


Figure 11 : CAN frame

- Start of frame

The start of frame (SOF) is a single symbol at dominant state.

- Message identifier

The message identifier is an 11 bits parameter freely managed by the application designers. A message identifier is assigned to an input/output or to a group of inputs/outputs

- Control bits

There are two controls bits, the remote transmission request (RTR) which indicate the frame type and the identifier extension (IDE) which enable 29-bits message identifier instead of 11-bits.

- Data length code

Data length code (DLC) is a 4 bits parameter which specifies the length of the Data field. 4 bits allows 16 different values but only 9 permitted.

DLC	0	1	2	3	4	5	6	7	8
Length in bits	0	8	16	24	32	40	48	56	64

- Data field

The content of Data field can be freely managed by the developers but all actively receiving nodes must be capable to interpret its content.

- Cyclic redundancy check

The cyclic redundancy check (CRC) enables the receiver to detect transmission errors. The coefficients are generated modulo-2.

- First delimiter

One symbol time in order to let the receiver(s) check the CRC.

- Acknowledge

Acknowledge (ACK) is in dominant state if no error is detected otherwise in recessive state.

- End of frame

End of frame (EOF) is a 7 symbol field in recessive state.

- Inter-frame space

The time between two different frames is at least 2 symbol times which are in recessive state.

6.6.2 Power consumption

In order to work properly, an interface between the CAN protocol controller and the physical wires of the bus lines must be implemented. The CAN transceiver provides differential transmit capability to the bus and differential receive capability to the CAN controller.

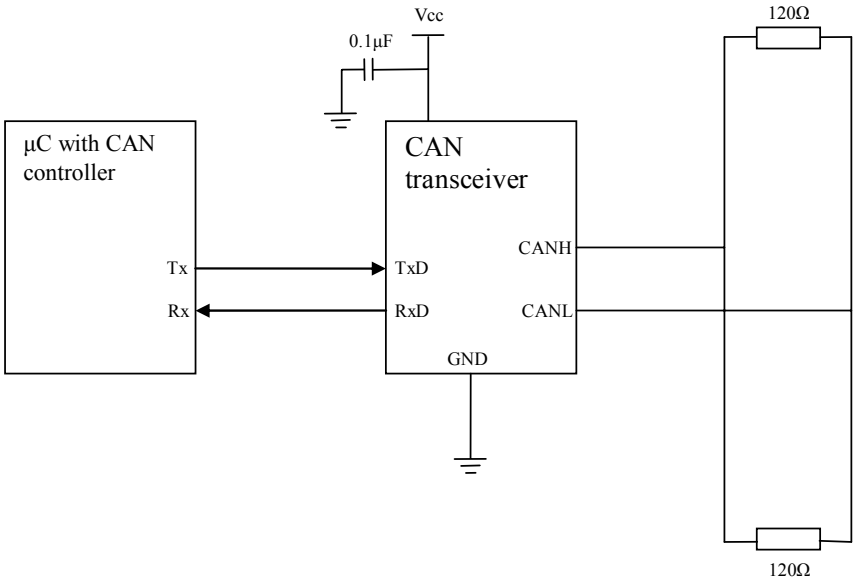


Figure 12 : Typical operating circuit

The MAX3051 CAN transceiver from MAXIM provides the features required for our use. It works with a typical supply voltage of 3.3[V] (value of our power bus) and a low supply current. It typically use 35[mA] (max: 70) to send a dominant bit and 2[mA] (max: 5) to send a recessive bit. Moreover, it provides a standby mode which only uses 8[µA] (max: 15). The transceiver use current only while speaking, and only one device can speak at a time. While there is nothing to send or while listening, the transceiver is in standby mode. The power consumption of the transceiver is slightly depending on data rate.

(This can transceiver is just given as example and will not provide the final architecture description)

As the final duty cycle on the bus can not be yet defined and in order to have an idea on the power consumption of the bus, we make the assumption that, each second, the CDMS sends one frame to all subsystems connected on the bus, and the subsystems send one frame to the CDMS. Moreover,

the size of frame is set to a mid-range value of 40 'dominants' bits. As at that time four subsystems are connected, and to let the highest priority Message Identifier for the ground station, the highest value of the message identifier will be 5; this value will be used to calculate the approximated power consumption. With those assumptions, the size of one CAN frame is 98 bits long and contain 63 dominant bits and 35 recessive bit. Including the minimal inter-frame space of 2 recessive bits, the final value is 100 bits long with, 63 dominant and 37 recessive. The calculation is done with typical power consumption of the MAX3051 CAN transceiver.

Two different data rate value will be used, one at 100[kbps] and the other at 1[Mbps]

- Legend

D	: *data rate*
T_{bit}	: *time to transmit one bit*
I_R / I_D	: *current used to transmit one recessive / dom inant bit*
P_R / P_D	: *power used to transmit one recessive / dom inant bit*
E_R / E_D	: *enery used to transmit one recessive / dom inant bit*
E_{frame}	: *energy used to transmit one frame*
E_{comm}	: *energy used to make one communication cycle*
$E_{onOneHour}$	: *energy used on one hour*

- Data rate = 100[kbps]

$$D = 100[kbps] \Rightarrow T_{bit} = \frac{1}{D} = 10[\mu s]$$
$$I_R = 2[mA] \Rightarrow P_R = 2[mA] \cdot 3{,}3[V] = 6{,}6[mW] \Rightarrow E_R = 6{,}6[mW] \cdot 10[\mu s] = 66[\eta J]$$
$$I_D = 35[mA] \Rightarrow P_D = 35[mA] \cdot 3{,}3[V] = 115.5[mW] \Rightarrow E_D = 115.5[mW] \cdot 10[\mu s] = 1{,}155[\mu J]$$
$$E_{frame} = 37 \cdot 66[\eta J] + 63 \cdot 1{,}155[\mu J] = 75{,}2[\mu J]$$
$$E_{comm} = 10 \cdot 75{,}2[\mu J] = 752[\mu J]$$
$$E_{onOneHour} = 752[\mu J] \cdot 3600 = 2.7[Wh]$$

- Data rate = 1 [Mbps]

$$D = 1[Mbps] \Rightarrow T_{bit} = \frac{1}{D} = 1[\mu s]$$
$$I_R = 2[mA] \Rightarrow P_R = 2[mA] \cdot 3{,}3[V] = 6{,}6[mW] \Rightarrow E_R = 6{,}6[mW] \cdot 1[\mu s] = 6{,}6[\eta J]$$
$$I_D = 35[mA] \Rightarrow P_D = 35[mA] \cdot 3{,}3[V] = 115{,}5[mW] \Rightarrow E_D = 115{,}5[mW] \cdot 1[\mu s] = 115{,}5[nJ]$$
$$E_{frame} = 37 \cdot 6{,}6[\eta J] + 63 \cdot 115{,}5[nJ] = 7{,}5[\mu J]$$
$$E_{comm} = 10 \cdot 7{,}5[\mu J] = 75[\mu J]$$
$$E_{onOneHour} = 75[\mu J] \cdot 3600 = 270[mWh]$$

As we can see with those calculations, the highest data rate is the lowest power consumption.

According to use the highest data rate of 1[Mbps], the only way to reduce power consumption is to reduce the number of frame transmitted, or to find another transceiver which use less power.

I remember that this power consumption is not the final value, but only an approximated value.

6.7 I²C bus how does it work?

This section will provide a rapid view of the I²C bus, in order to have a full specification of this bus, please refer to [N3]

The I²C bus (Inter-IC Bus) or (IIC Bus) was originally designed to be a battery control interface. The I²C bus uses a bi-directional Serial Clock Line (SCL) and Serial Data Line (SDL). Both lines are pulled high via a resistor (R_p). Resistor R_s is optional, and used for ESD protection for "Hot-Swap" devices. No other lines are specified. Three speed modes are specified: Standard; 100[kbps] (bits per second), Fast mode; 400[kbps], High speed mode; 3,4[Mbps]. I²C, due to its two-wire nature (one clock, one data) can only communicate half-duplex. The maximum bus capacitance is 400[pF], which sets the maximum number of devices on the bus and the maximum line length. The interface uses 8 bit long bytes, MSB (Most Significant Bit) first, which each device having a unique address. Any device may be a transmitter or receiver, and a master or slave. Data and clock are set from the master, valid while the clock line is high. The link may have multiple masters and slaves on the bus, bit only one master may be active at any time. Slaves may receive or transmit data to the master. I²C defines the electrical layer and protocol, and was developed by Philips Semiconductor. V_{dd} may be different for each device, but all devices have to relate their output levels to the voltage produces by the pull-up resistor (R_p)

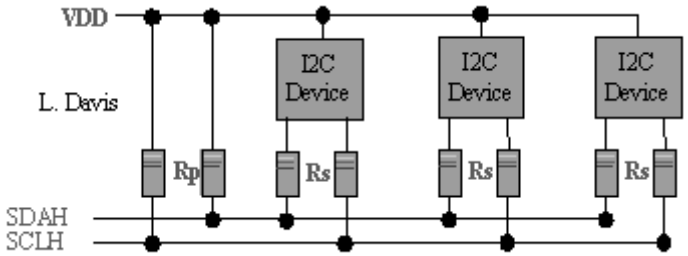


Figure 13 : I²C bus topology

6.8 Main bus choice

The bus that will be used on the Swisscube is the IIC; this bus is widely used on other cubesat and provides the lowest power consumption.

7 CDMS SPECIFICATION

7.1 CDMS Functional Specification

The CDMS board realizes the processing and scheduling functions of the Swisscube satellite. The mains activities of the CDMS are:

- To perform the scheduling of the Swisscube functions
- To perform the data storage of the Swisscube
- To execute the ADCS algorithm
- To give the reference time aboard the Swisscube
- To maintain life statuses of the Swisscube
- To realize some specifics treatment on data coming out of sensors

7.1.1 CDMS modes

The CDMS shall support the following modes:

- **Off**: CDMS is off, no function is available, no power consumption
- **Initialization**: after a power on, transient mode in which initialization of functions are performed
- **Standby**: CDMS is in low power mode, only reference time function is active
- **Operational**: CDMS is fully operational
- **Shutdown**: after a power off, transient mode which led to off mode

7.1.2 Scheduling of the Swisscube functions

The CDMS shall be able to perform the scheduling of the Swisscube functions. In order to fulfill this function the CDMS shall implement a real time clock and an operational system based on ECOS or RTEMS.

7.1.3 CDMS memories (program and data storage)

Final boot program shall be stored in ROM. Flash memories shall be used for application SW and data storage. RAM shall be used for execution program memory and temporary data storage.

The capacities of the memories are given in the following table.

Memory type	Capacity [Bytes]
ROM	512k
Flash	4M
RAM	512k

The CDMS shall manage the following two data types:

- data coming from the payload
- telemetry and command of the Swisscube

7.1.4 ACDS algorithm implementation

The CDMS shall be able to execute the ACDS algorithm on 32 bit data by means of a HW multiplier.

7.1.5 Time reference generation

The CDMS shall generate a time clock reference. The format shall be compatible with: year, month, day, hour, min and sec.

7.1.6 CDMS status

The CDMS shall be able to deliver the following statuses:

- Board T°
- Reset source
- Mode status

7.1.7 Payload data treatment

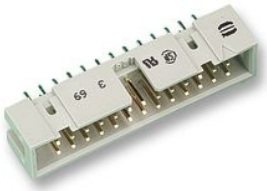
The CDMS shall be able to timestamp each image coming from the payload.

7.2 CDMS Interface requirements

In order to realize these functions the CDMS board interfaces with the following subsystems by means of the following links:

- To realize some specifics treatment on data coming out of sensors
- one power link composed of GND and 3.3V signals
- one I2C communication bus
- one external wakeup signal
- one external boot program selection
- miscellaneous I/O signals (4 analogue inputs, 16 digital I/O, test)
- one JTAG test debug interface

The following table gives the connectors definition of the CDMS board.

IF link	Connector type	PIN attribution
I2C and Subsystem	Header 10 	1:GND 2:SDL 3:GND 4:SCL 5:GND 6:MCK 7:GND 8:BOOT_SELECT 9:WAKEUP 10:NC
Power	Law voltage connector 	1:GND 2:Vdd
Miscellaneous	Header 24	1:Vdd 2:VddBU 3 - 6: AD2 - AD5 7 - 15: PA0 - PA8 16 - 23 :PB8 - PB15 24:GND
Test	Header 10 	1:Vdd 2:NC 3:TMS 4:NC 5:TCK 6:NJT_RST 7:TDO 8:NREST 9:TDI 10:GND

7.3 CDMS Electrical requirements

7.3.1 Power consumption

- Standby mode: 1mW
- Operational mode: 250 mW peak, 150 mW average

7.3.2 Power ON/OFF

The power on of the CDMS is directly performed with 3.3V bus.

The power off is given by a specific I2C command.

7.3.3 Input voltage range

The nominal voltage is: from 3V to 3.6V. In case of perturbation the voltage can be increased up to 4.2V.

7.3.4 Wakeup command

A specific external signal cans wakeup the CDMS board (standby mode → operational mode).

7.3.5 I2C

For I2C bus electrical interfaces refer to [N1].

7.4 CDMS Mechanical requirements

The CDMS board shall fulfill the following dimensions:

- Width: 40[mm]
- Length: 90[mm]
- Height: 10[mm]

The maximum weight of the CDMS shall be: 90 g. No requirement is given for the CoG.

7.5 CDMS environment constraints

7.5.1 RADIATIONS

- **Total dose**

  The table hereafter summarizes the trapped radiation dose cumulated for 3 months and 1 year behind 100 mil of Aluminium (2.5 mm) and behind 40 mils of Al (1mm). The analysis assumes a spherical shell shield configuration. The analysis was done using the ESA Spenvis Tool. It assumes solar maxima.

Configuration	Cumulated Dose (krad[Si], 2.5 mm Al)	Cumulated Dose (krad[Si], 1 mm Al)
400 km 3 months	0.26	1.1
400 km 1 year	1.0	4.6
1000 km 3 months	2.1	9.2
1000 km 1 year	8.7	37.4

Table III-1: Trapped radiation dose worst and best cases.

According to the orbit of the Swisscube the CDMS shall support a TID of maximum 1kRAD.

- **SEU**

  SEUs can be mitigated by hardware or software design practices for critical function if necessary.

- **SEL**

  A separate latch-up protection circuit shall be implemented on the CDMS board.

7.5.2 TEMPERATURE

- **Non-operational**

  In OFF mode the CDMS shall support T° range from -40°C to 125°C.

- **Operational**

  In operational mode the CDMS shall support T° range from -10°C to 50°C.

7.5.3 Vibration

The CDMS shall support during the launch an acceleration of 10g and a vibration frequency of 100Hz along the Z axis.

7.5.4 LIFE TIME

The total life time of the CDMS shall be 2 years. The operational life time shall be 1 year on earth and 3 months in orbit.

7.6 TESTS

The CDMS shall be accessible by specific interfaces for test purpose.

7.7 CRITICAL POINTS

Specific attention shall be paid on power consumption of the CDMS. Off-the-shell components used for the CDMS must be chosen according to environmental conditions specified in chapter 5.5.

8 HARDWARE DEVELOPMENT

This section will speak about the hardware development of the CDMS, from the first functional architecture derived from the specification to the board layout that will be manufactured.

8.1 Functional Architecture

Conforming to the specification of the CDMS presented on section 5, the following functional architecture has been drawn.

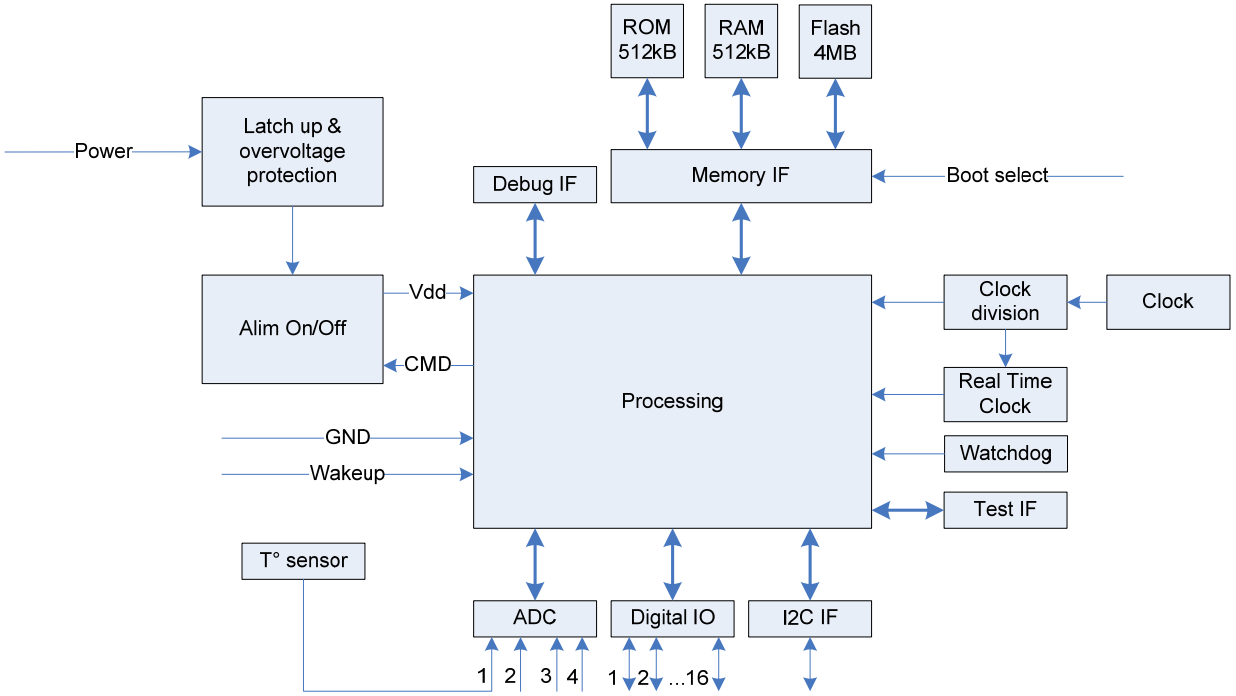


Figure 14 : Functional architecture

The one thing probably that limits the design of the architecture the most, is the space-operating demand. As no one will be able to reset the processor manually in case of a software malfunction, the system must be able to do this by itself. The way is done is to use a watchdog timer. The timer is increased on a regular basis (typically it is clocked by the master clock) and when it reaches a certain value, it resets the processor. Of course this is not what we want when the computer is working correctly. Therefore a part of the software must reset the watchdog before it reaches the reset value. This id done when the software is working correctly, but if it gets in a deadlock the watchdog is not reset, and therefore it resets the system. The watchdog can either be build-in in the processor or it can be a separate unit as shown in the figure 3.

When studying the figure you will find three different types of memory: ROM, flash, and RAM. These three types are also chosen because of the space-operating nature of this computer. In space, the radiation may lead to bit-flips in memory after which the software may do unpredictable things. The watchdog will obviously reset the processor if this happens, and the processor starts loading the boot-software. It is essential that this software always work correctly, and therefore it must be stored

in a memory, where bit-flips do not occur. Some types of ROM (read only memory) have this feature.

The flash will be used to store the operating system and other software. Some of this software could be store in ROM, but as it might be necessary to change parts of it, when the Swisscube is in space, it will be stores in flash. On the issue of flash memory ESA experts tend to give the advice from do not use to no problem at all. However XI-V (a cubesat developed by the university of Tokyo) team has used it on their satellite (doing prior radiation testing) with success.

The RAM (random access memory) will be used as a temporary memory when the programs run, and as a place where measured values can be stored.

In order to communicate with other elements, the computer must provide different interface, a bus interface to exchange data with other subsystem; the bus selection has been presented in section 4, both analogue and digital interfaces. Since some processors may not be able to supply very much current it might be necessary to insert drivers between the digital interface and the processor.

Depending on which processor is chosen we may have to add a real time clock (RTC) to the board, but if the processor has enough timers, this can also be implemented in software. The purpose of the real time clock is to make it possible to schedule tasks.

As we described above, the CMOS component are sensible to latch-up. To avoid burn-out, a latch-up protection circuitry must be implemented between power bus and sensible component. When a latch-up occurs the power must be turn off immediately.

One last thing of special interest is shown on the figure: The debug interface. The purpose of this is to make it possible to find and correct errors in the hard- and software design. It also makes it possible to upload new software to the flash. It consists of measuring points for important signals and some sort of interface to the processor and the flash.

8.2 Components Selection

Satellites are normally extremely expensive and account for years of development. To improve the odds for success, it is custom to choose components of great reliability for satellites. To be absolutely sure of reliability, it is not uncommon to buy components that have actually been tested in space. This form of certainty is very expensive and lead times are a.

A cheaper alternative is to choose a Commercial off-the-shelf (COTS) component that has been used in space before. Another component just like it will probably work in space too. The only problem with this alternative is, that development of satellites is a long process, and when you find a component that has been successfully used in space, it is most likely more than two or three years old. In these days, where new and better technologies revolutionize the chip-marked every other day, it is very tempting to choose new and more powerful components for your design. We have agreed to yield to this temptation. Furthermore we will perform tests to verify that the components do not collapse with the first signs of space radiation.

8.2.1 Microcontroller

We need a processor for the different subsystem that can execute the onboard programs in a reasonable way. The compulsory tasks of the satellite (power control, communication, attitude control, etc.) could be managed by a quite humble processor or microcontroller. Processors and microcontroller from a few years back would do the job with no problems and we could choose a very reliable one.

Nevertheless, we examined both new and old processors because of the above argumentation on why to take a chance on newer products.

We found that the newer processors have made drastic improvements on power consumption even though the speed of the processors has greatly increased.

8.2.1.1 Analysis criteria

In order to choose the perfect (or at least the best) processor, we have evaluated the different processors in different categories and summed up the information in the table below. The different categories have different weights, as they are not equally important. In some categories, the rating has been made from a reference value, which we have set from average specification of the component. The categories are:

- Peak power consumption (1 MIPS)

As we only have a few Watts available to all subsystems in the satellite, it is of outmost importance that the processor has low power consumption. The 1 MIPS value is not the final frequency of the processor, but the value which the power consumption of processors will be compared to.

Power consumption has the weight of 5.

- Standby power consumption

During the scheduling of the mission, some subsystems will not be used. In order to save power those subsystems will be in standby mode. The reference value has been set at 3[mW].

Standby consumption has the weight of 3.

- Temperature Range

The temperature will differ significantly depending on the location of the satellite. In the sunny side of earth, the surface facing the sun will be very warm and in the shadow of earth, everything will get very cold. The temperature range has been set to 0°C to 40°C for normal operation. But in special case the temperature might exceed this range in both directions, which is why we would like to have components with great temperature tolerances. Luckily, it is normal for military/industrial components to be operational from -40°C to 80°C, which we believe is sufficient.

Temperature Range has the weight of 3.

- Already used in space

As the name says, by 'already used in space', we mean whether or not the processor has been tried in space before, and if this processor has work properly.

Space rating has a weight of 4.

- Power voltage (core & I/O)

The power bus will provide 3.3[V]. Instead of having numerous supply voltages, which take place and weight, it is preferable to run a single supply voltage for the core and I/O of the processor. This standard supply voltage can also be used for other parts of the computer.

Those two categories have the weight of 4.

- Package

The place available on the satellite is very limited too. We look for having components where the pins are located along the edges, which offers greater mechanical characteristic than ball grid array components where the pins are distributed underneath the chip. Due to the nature of project, it should be interesting to have ceramic packaging than usual plastic.

Package has the weight of 3.

- I/O compatibility

As the sensors are not yet defined we made the assumption that 5[V] tolerant I/O which is a standard voltage will be appreciable.

The I/O compatibility has the weight of 2.

- MIPS

We made the assumption that a microcontroller which provides between 10 to 20 MIPS will be sufficient to make the processing of the ADCS. The reference value for 8 bit microcontroller is set to 1 to 10 MIPS.

This criterion has the weight of 4.

- Multiplier

As some processing will require multiplication, it should be interesting to have an integrated multiplier, but not required.

The multiplier has the weight of 1.

- Digital and Analogical I/O number

All sensors and actuators need access to the processor via the I/O pins of the processor, it is preferable to have sufficient I/O pin in order to connect all the peripherals.

We made the assumption that 12 digital I/O and 8 A/D converters will be sufficient.

Those criteria have the weight of 3.

- UART

The UART component will be used to program the chip on the earth during the debug and test phase, and probably for the survival and science bus during spatial mission.

UART has the weight of 2.

- Development environment

In order to have a fully functional microcontroller software must be developed and implemented. This criterion compares the facilities of development environment provided by the manufacturer. It is really important to have powerful tools to manage the hardware in order to debug software.

Development environment has the weight of 4.

8.2.1.2 The small microcontroller

Items		Peak Power consumption (1 MIPS)	Standby Power consumption	Temperature range	Package	Power voltage core	Power voltage I/O	I/O compatibility	Already used in space	MIPS	Multiplier	Digital I/O number	Analog I/O number	UART	development environment	Results
Reference		20mW	3mW	-40°C to + 85°C	ceramic/flat < 5cm2/ <5gr.	3.3V	3.3V	5V tolerant	minimum one success	10 to 20	1	12	8	1	Facilities	
weighting factor		5	3	3	3	4	4	2	3	3	1	3	3	2	4	
Items	TI MSP 430	++	++	o	++	o	o	o	++	+	++	+	+	o	++	176
	Infineon C161	--	-	o	+	o	o	o	++	-	-	++	+	o	--	122
	PIC18LF4680	+	++	o	++	o	o	o	-	o	+	++	+	o	++	161
	PIC18F4680	+	++	o	++	--	--	-	-	o	+	++	+	o	++	143
	PIC18LF8680	o	++	+	++	o	o	o	-	o	+	++	++	o	++	162
	ATMEL AT90CAN128	o	++	o	+	o	o	o	-	+	o	++	o	o	++	152

With this table we see that the TI MSP430 series offers the best performance for our application, this microcontroller is used on the cubesat Kit and offer 16-bit RISC performance for ultra low power consumption. This microcontroller will not be discussed later as the CDMS will be powered by a 32-bit microcontroller.

8.2.1.3 The 32-bit microcontroller

This is not an exhaustive list, but a list that compare the main core family which would be interesting in term of low power consumption and features.

Items		Peak Power consumption (1 MIPS)	Standby Power consumption	Temperature range	Package	Power voltage core	Power voltage I/O	I/O compatibility	Already used in space	MIPS	Multiplier	Digital I/O number	Analog I/O number	UART	development environment	Results
Reference		80mW	3mW	-40°C to + 85°C	ceramic/flat < 5cm2/ <5gr.	3.3V	3.3V	5V tolerant	minimum one success	10 to 20	1	12	8	1	Facilities	
weighting factor		5	3	3	3	4	4	2	3	3	1	3	3	2	4	
Items	NEC UPD703286 (V850ES/SJ2)	++	o	o	-	o	o	o	-	+	o	++	+	++	--	141
	FUJITSU MB91F267N (FR60 lite)	--	-	+	-	--	--	o	-	+	-	++	+	++	--	106
	STM STR910FM32X (ARM966)	+	o	o	-	--	o	o	-	++	-	++	o	++	+	139
	Motorola Dragonball VZ	--	-	o	-	o	o	-	+	-	-	+	-	++	--	104
	ATMEL AT91M55800A (ARM7TDMI)	+	++	o	-	o	o	o	-	+	-	++	o	++	++	154
	ATMEL AT91SAM7A1 (ARM7TDMI)	+	++	o	-	o	o	o	o	+	-	++	o	++	++	157

As shown on this table the best microcontroller is one from ATMEL based on an ARM7TDMI core. The ARM7TDMI core is a 32-bit embedded RISC processor delivered as a hard macrocell optimized to provide the best combination of performance, power and area characteristics. The ARM7TDMI core enables system designers to build embedded devices requiring small size, low power and high performance. The AT91SAM7A1 has appreciatively the same results as the AT91M55800A. Those two microcontrollers have been compared more in details, and the final selection is the AT91M55800A because this microcontroller has an internal real time clock and a more accurate power management controller.

8.2.2 Memories

The CDMS must provide sufficient memory to store flight software, science data, telemetry and enough RAM to execute all the software. Different type of memory exists; they can be classed in the three following categories.

- *Volatile memory* requires constant power to maintain the stored information. The SRAM (static random access memory) is the most known example.
- *Dynamic memory* is volatile memory which also requires that stored information is periodically refreshed, or read and rewritten without modifications. One example is the DRAM (dynamic random access memory)
- *Non-volatile memory* will retain the stored information even if it is not constantly supplied with electrical power. FLASH and EEPROM (electrically erasable programmable read-only memory) are two different widely used non-volatile memory.

To compare the possible choice of memory we looked at the following categories:

- Size

As say before a size of two times 16[Mbit] will suit our needs, a 16bit data bus is preferred.

- Supply voltage

A supply voltage of 3.3[V] is preferred as this is the supply voltage of the processor.

- Read/write/stand-by current

Read and write current are the current drawn during active time, stand-by current is the current drawn when the component is passive.

- Write cycle

This category is only shown for non-volatile memory and present the average number of possible write cycle before the memory encounter failure.

8.2.2.1 Data Storage memory

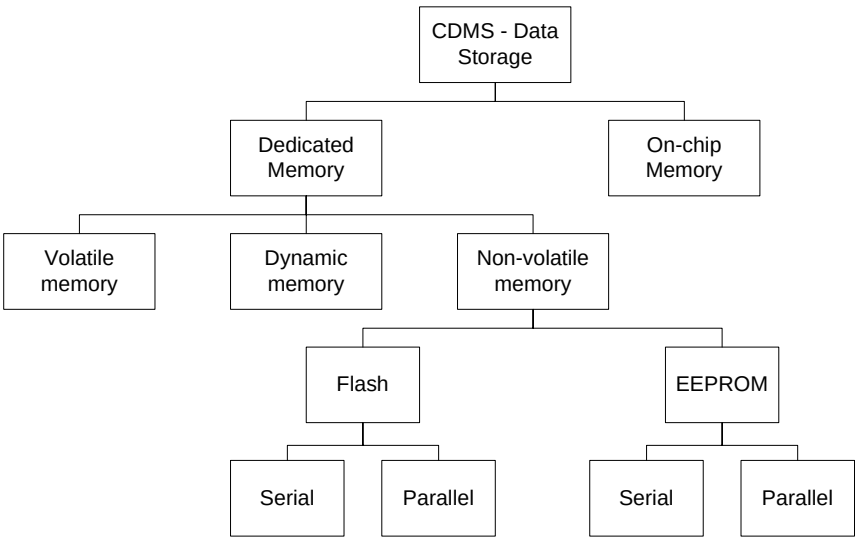


Figure 15 : Memory trade-off

In order to save images captured by the payload, housekeeping, telemetry and command, the CDMS board should provide a 32[Mbit] memory.

It could be interesting to keep data either if the subsystem encounters a dysfunction and must be shut-down or reboot, in that way non-volatile memory will be used. EEPROM has similar qualities than Flash but differs in the way that with EEPROM it is possible to rewrite a single byte at any address. It is a big advantage compared to a Flash where you have to erase an entire sector at a time. The sector size differs in function of the type of Flash selected. The Flash and EEPROM memory chip should be either access in serial or parallel. At first approximation it is not possible to compare reliability between Flash and EEPROM for a spatial use. However, FLASH is used in many satellites including the Danish Ørsted satellite which has been in nominal mode for 6 years now in LEO. AMSAT also use FLASH. SSTL use FLASH. AAU is now using FLASH, too. Moreover FLASH usually provides bigger memory size.

The CDMS is based upon a 32-bit microcontroller; the AT91SAM7A1 which offer a dedicated memory interface, in that way a parallel memory will present the best choice.

For comparison a space radiation approved parallel memory chip is the Maxwell 79LV0408. This memory chip has a size of 4[Mbit] and has a power dissipation of 88[mW/MHz] in active mode and 440[µW] in stand-by mode. The supply voltage used is 3.3[V] and the size is 22*25*6[mm] for a weight of 10[g].

Not all the memory we look for are presented on this table, only the most interesting one are represented.

Manufacturer	Item	Type	Write cycle	Supply voltage	Read Current	Write current	Size	Density
AMD	AM29LV017D	FLASH	1M	3.3	9mA/5Mhz	15mA/5Mhz	20*10*1.2	2M*8
AMD	AM29LV320	FLASH	1M	3.3	9mA/5Mhz	15mA/5Mhz	20*10*1.2	2M*16
AMD Spansion	S29AL016D	FLASH	1M	3.3	7mA/5Mhz	15mA/5Mhz	20*10*1.2	1M*16
ST	M29W320D	FLASH	100k	3.3	10mA/5Mhz	20mA/5Mhz	20*10*1.2	2M*16
ATMEL	AT27C040	EEPROM	100k	3.3	50mA/5Mhz	50mA/5Mhz	20*10*1.2	512k*8
ATMEL	AT28LV010	EEPROM	100k	3.3	15mA/5Mhz	15mA/5Mhz	20*8*1.2	128k*8

The DTUsat use AM29LV017D and AAUsat use AM29LV320 FLASH memory from AMD. Those memories provide either low power consumption and moreover a really high endurance, however those products have been retired and are not available for designs. This chip family is superseding by the S29AL family which will suit our needs. The S29AL016D provides 16Mbit and a 16-bit data bus.

It has been decided to implement two different memory chips; one will be used in order to save science data, the other to save command, housekeeping, and telemetry.

8.2.2.2 Volatile memory

The volatile memory will be used as working memory, for the stack and to save temporary data too. The AT91M55800A provides only 64[kbit] of internal RAM, which is probably not enough. At that time we have no information from the Flight software development group about the amount of volatile memory required. Comparing to other Cubesat and some information about operating system that are optimized for our requirement, it seems that 1 to 4[Mbit] will be sufficient, but this value must be confirmed. In the case of our project SRAM is preferred as other volatile memory because the power consumption will be smaller.

Manufacturer	Items	active current (1Mhz)	Standby current	Temp. range	power supply	Speed	Size	Density
nanoAmp	N04L163WC1A	2mA	4uA	-40 +85	2.3 - 3.6	70ns	12*18*1.1 TQFP	256k * 16
Samsung	K6F4016U6G	4mA	3uA	-40 +85	2.7-3.3	55/70ns	6*7*1 BGA	256k * 16
Renesas	R1LV0416CSB-7LI	2mA	2uA	-40 +85	2.2 - 3.6	70ns	12*19*1.2 TQFP	256k * 16
Nec	uPD444012A-B	6mA	6uA	-40 +85	2.7-3.6	70ns	12*18*1.2 TQFP	256k * 16
Cypress	CY62147DV30-55ZSXE	1.5mA	2uA	-40 +125	2.2 - 3.6	55ns	12*18*1.2 TSOP	256k * 16
Cypress	CY62126DV30-55ZSE	0.85mA	2uA	-40 +125	2.2 - 3.6	55ns	12*18*1.2 TSOP	64k * 16

On this table are compared some proposition for the working memory. Two different memory chips offer the best performance, the CY621xxDV30 family from Cypress and the R1LV0416 from Renesas. The DTUsat has also used memory from Cypress, the CY62162DV30 which is no more presented on the Cypress website. Due to the difficulty to find Cypress memory chip it has been decided to use the one from Renesas which provides appreciatively the same performances.

8.2.2.3 Flight Software memory

As the spatial environment is particular, it is planned to write the final flight software on a ROM. The ROM is a one-time memory chip, for this reason it is quite impossible to modify the value of stored data. As the ROM will be programmed only with the final software, the selection is not done yet. Since the first board we have to make is a test board that will be used as a development tool for the satellite, it is not convenient with a ROM that only can be programmed once. Therefore we have decided to use a flash component to simulate the boot ROM. To avoid making more drivers than necessary we will use a flash of the same kind as the flash for the data storage. As the AT91M55800A is only able to boot from the memory selected by the chip select 0; a jumper will be accessible on the CDMS board allowing selecting the ROM or Flash memory chip. The ROM selected is the AT27BV4096 which provide 4[Mbit] with a 16-bit data bus. The typical current required for reading the memory is only 6[mA/5Mhz] for a single power supply of 3.3[V].

8.2.3 Oscillator

The majority of clock sources for microcontroller can be grouped into two types: those based on mechanical resonant devices, such as crystals and ceramic resonators, and those based on electrical phase-shift circuits such as RC (resistor, capacitor) oscillators. Silicon oscillators are typically a fully integrated version of the RC oscillator with the added benefits of current sources, matched resistors and capacitors, and temperature compensation circuits for increased stability. Two examples of clock source are illustrated in the following figure. Figure (a) shows a Pierce oscillator configuration suitable for use with mechanical resonant device like crystals and ceramic resonators, while figure (b) shows a simple RC feedback oscillator.

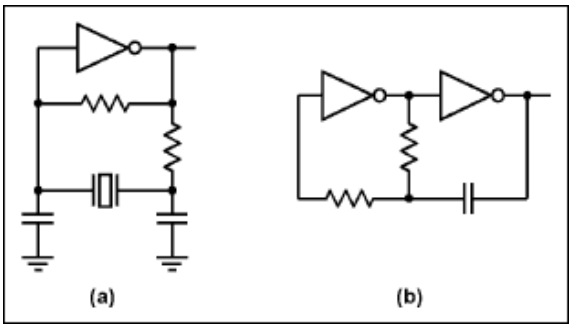


Figure 16 : (a) pierce oscillator (b) RC oscillator

- Primary differences between mechanical resonator and RC oscillators

Crystals and ceramic resonator-based oscillators (mechanical) typically provide very high initial accuracy and a moderately low temperature coefficient. RC oscillators, in contrast, provide fast start-up and low cost, but generally suffer from poor accuracy over temperature and supply voltage, and show variation from 5% to 50% of nominal output frequency. While the circuits illustrated above can produce clean reliable clock signals, their performance will be heavily influenced by environmental conditions, circuit component choice, and the layout of the oscillator circuit. Ceramic resonators and their associated load capacitance values must be optimized for operation with particular logic families. Crystals are not as sensitive to amplifier selection but are susceptible to frequency shifts (and even damage) when overdriven. Environmental factors like electromagnetic interference (EMI), mechanical vibration and shock, humidity, and temperature affect oscillator operation. These environmental factors can cause output frequency changes, increased jitter, and in severe cases, can cause the oscillator to stop functioning.

- Oscillator modules

Many of the considerations described above can be avoided through use of oscillator modules. These modules contain all oscillator circuit components and provide a clock signal as a low-impedance square-wave output. Operation is guaranteed over a range of conditions. Crystal oscillator modules and fully integrated silicon oscillators are most common. Crystal oscillators are more precise than discrete component RC oscillator circuits, and many provide comparable accuracy to ceramic resonator-based oscillators.

- Power consumption

Power consumption is another important consideration of oscillator selection. The power consumption of discrete component crystal-oscillator circuits is primarily determined by the feedback-amplifier supply current and by the in-circuit capacitance values used. The power consumption of amplifiers fabricated in CMOS is largely proportional to the operating frequency and can be expressed as a power-dissipation capacitance value.

The different power consumptions below are given for example.

The power-dissipation capacitance value of an HC04 inverter gates used as an inverting amplifier is typically 90[pF]. For operating at 4[MHz] from a 5[V] supply, this equates to a supply current of 1.8[mA] The discrete component crystal oscillator circuit will typically include an additional load capacitance value of 20[pF], and the total supply current becomes 2.2[mA].

Ceramic resonator circuits typically specify larger load capacitance values than crystal circuits, and draw still more current than the crystal circuit using the same amplifier.

By comparison, crystal oscillator modules typically draw between 10[mA] and 60[mA] of supply current because o the temperature compensation and control functions included.

The supply current for silicon oscillators depends on type and function, and can range from a few [µA] for low-frequency (fixed) devices to ten of [mA] for programmable-frequency parts. A low-power silicon oscillator as the MAX7375 from Maxim draws less than 2[mA] when operating a 4[MHz].

8.2.3.1 Summary

Clock Source	Accuracy	Advantages	Disadvantages
Crystal	Medium to high	Low cost	Sensitive to EMI, vibration, and humidity. Complex circuit impedance matching.
Crystal Oscillator Module	Medium to high	Insensitive to EMI and humidity. No additional components or matching issues.	High cost; high power consumption; sensitive to vibration; large packaging.
Ceramic Resonator	Medium	Lower cost	Sensitive to EMI, vibration, and humidity.
Integrated Silicon Oscillator	Low to medium	Insensitive to EMI, vibration, and humidity. Fast startup, small size, and no additional components or matching issues.	Temperature sensitivity is generally worse than crystal and ceramic resonator types; high supply current with some types.
RC Oscillator	Very low	Lowest cost	Usually sensitive to EMI and humidity. Poor temperature and supply-voltage rejection performance.

Figure 17 : Oscillator comparison

As shown on the summary, an integrated silicon oscillator will provide the best choice for our need. This type of oscillator is insensitive to EMI and vibration, more of that it provides the ability to drive many microcontrollers. Typically, the MAX7375 is able to drive a load of about 100[pF] capacitance with an acceptable rise time, and a microcontroller typically has a 10[pF] fan-in capacitance.

8.2.3.2 Clock distribution trade-off

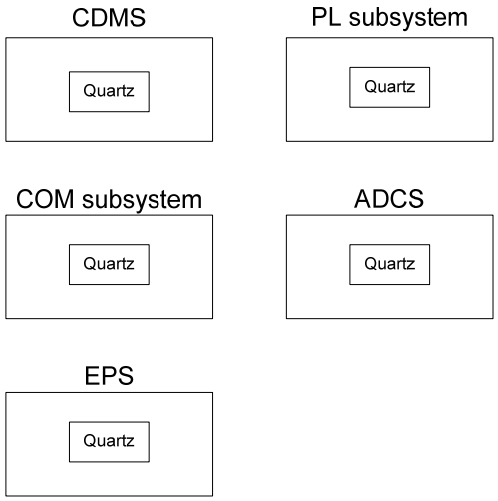


Figure 18 a)

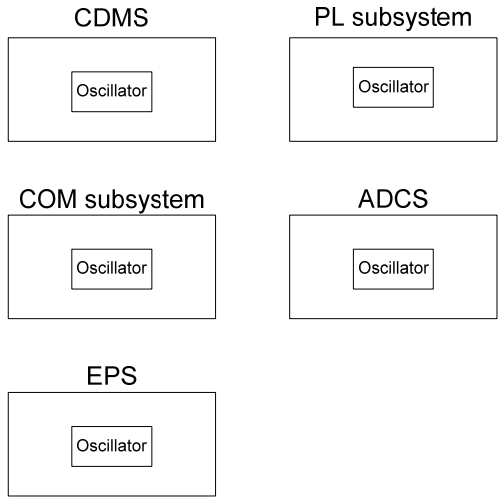


Figure 18 b)

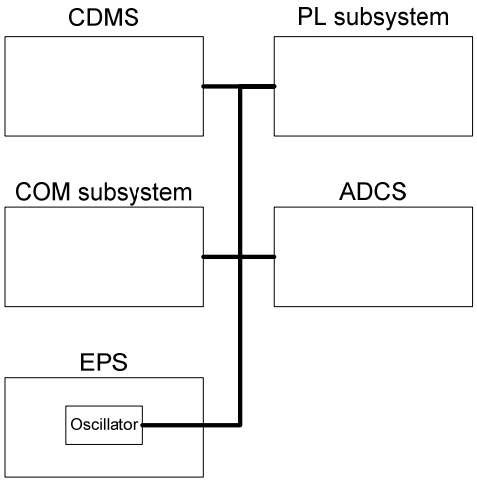


Figure 18 c)

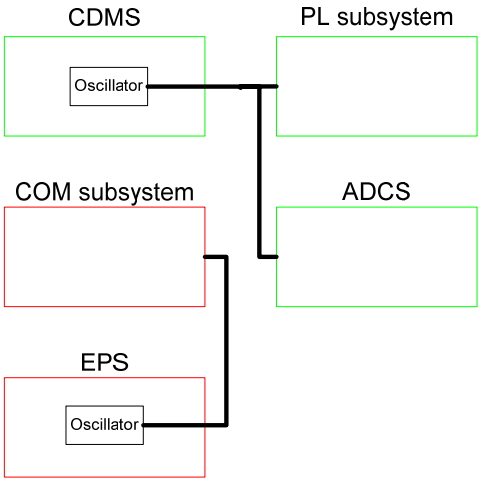


Figure 18 d)

The four figures showed further present the possibilities to implement the clock distribution. On the figure a) the clock is implemented with quartz and each subsystem has its own clock, this solution presents the lowest power consumption but, in term of reliability, this proposition is not the best one. It is quite difficult to have a good oscillation with a simple quartz assembly; quartz requires complex circuit impedance matching and is highly sensitive to EMI, vibration and humidity. In order to improve this reliability the figure b) was drawn. In this proposition, the quartz are now oscillator, the main problem is that oscillator consume a lot of power, and not enough power is able on the satellite to have five different oscillator. The figure c) presents the system based on one unique global oscillator. In the case of one subsystem does not work properly, it is possible that the problem goes until this unique oscillator and all the satellite is not able to work. The figure d) is probably the best solution. In this case the critical subsystems and the other are separated. The critical subsystem are presented in red on the figure, they are the COM subsystem and the EPS. Those two subsystems share one oscillator. In the case of EPS does not work, there is no power for other subsystem and the satellite is not able to function. On the other hand, if it is impossible to establish communication between ground station and satellite, due to COM dysfunction, the satellite is useless. The subsystems which are not essential for the basis operation of the satellite are presented in green. The CDMS, Payload and ADCS have one oscillator for the clock distribution.

The solution presented on the figure d) present the best compromise between power consumption and reliability. The clock distribution has not been decided yet, and the CDMS prototype board will offer the possibility to have the clock signal as an external signal.

8.3 AT91M55800A presentation

As the microcontroller is probably the most important component of the card it will be presented in details. This section will speak about the possibilities offered by this microcontroller.

The AT91M55800A is a member of the Atmel AT91 16/32-bit microcontroller family, which is based on the ARM7TDMI processor core. This processor has a high-performance 32-bit RISC architecture with a high-density 16-bit instruction set and very low power consumption. In addition, a large number of internally banked registers result in very fast exception handling, making the device ideal for real-time control applications. The fully programmable External Bus Interface

provides a direct connection to off-chip memory in as fast as one clock cycle for a read or write operation. An eight-level priority vectored interrupt controller in conjunction with the peripheral data controller significantly improves the real-time performance of the device. The device is manufactured using Atmel's high-density CMOS technology. By combining the ARM7TDMI processor core with an on-chip SRAM, a wide range of peripheral functions, analog interfaces and low-power oscillators on a monolithic chip, the Atmel AT91M55800A is a powerful microcontroller that provides a highly-flexible and cost-effective solution to many ultra low-power applications.

The AT91M55800A features:

- Utilizes the ARM7TDMI ARM Thumb Processor Core (presented on Appendix B)
- 8KBytes internal SRAM
- Fully-programmable External Bus Interface (EBI)
- 8-level priority, individually maskable, vectored interrupt controller
- Fifty-eight programmable I/O lines
- 6-channel 16-bit Timer/Counter
- Three USART
- Master/Slave SPI interface
- Programmable Watchdog timer
- 8-channel 10-bit ADC
- 2-channel 10-bit DAC
- Real-time Clock
- Battery backup operation and external alarm
- Advanced Power Management Controller (APMC)
- IEEE 1149.1 JTAG Boundary-scan on all digital pins
- Fully static operation: 0Hz to 33MHz

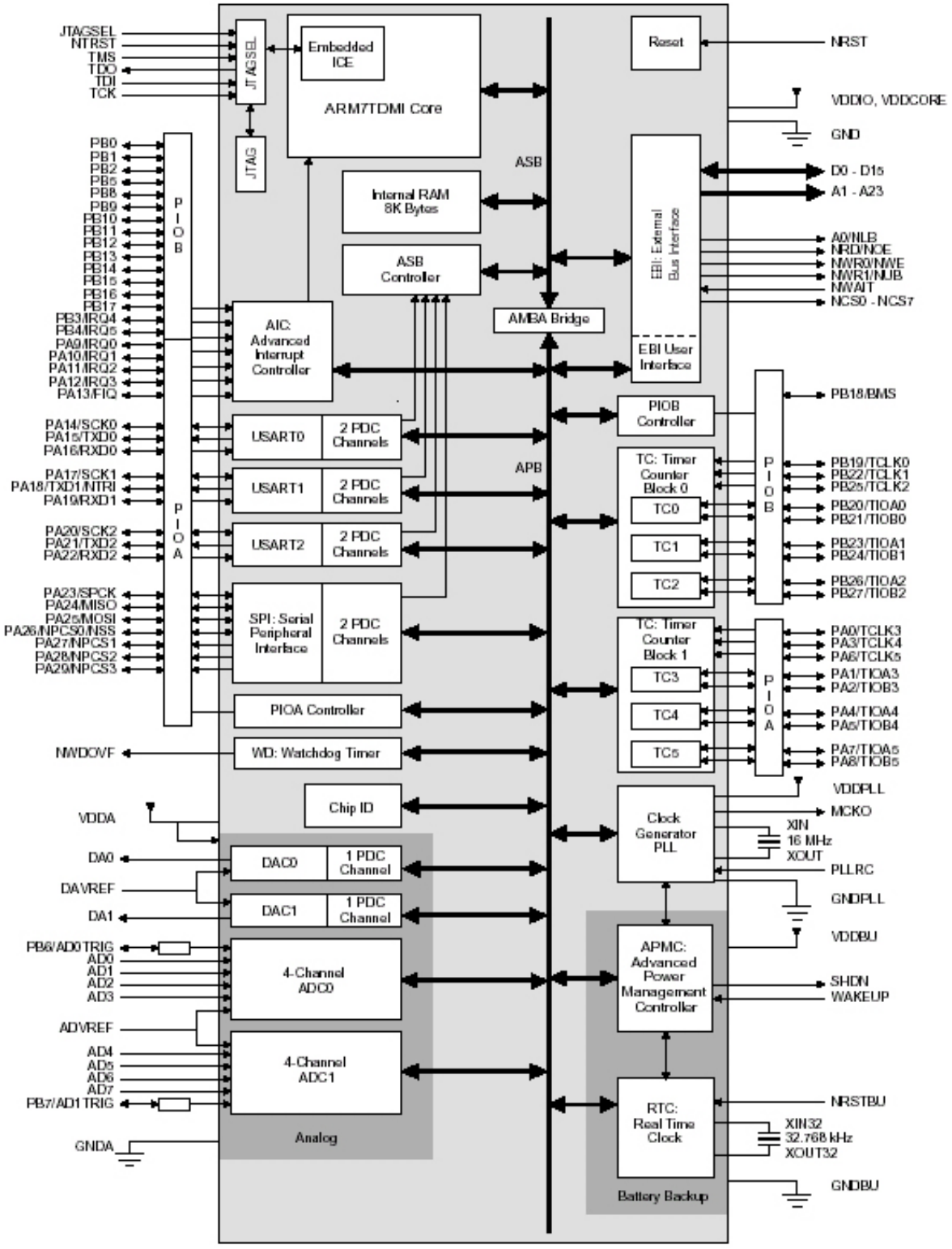


Figure 19 : AT91M55800A block diagram

8.3.1 JTAG

JTAG, an acronym for Join Test Action Group, is the usual name used for the IEEE 1149.1 standard entitled Standard Test Access Port and Boundary-Scan Architecture for test access ports used for testing printed circuit boards using boundary scan.

JTAG was standardized in 1990 as the IEEE std. 1149.1-1990. In 1994, a supplement that contains a description of the boundary scan description language (BSDL) was added. Since then, this standard has been adopted by electronics companies all over the world. Boundary-scan is nowadays mostly synonymous with JTAG.

While designed for printed circuit boards, it is primarily used for testing sub-blocks of integrated circuits, and is also useful as a mechanism for debugging embedded systems, providing a convenient "back door" into the system. When used as a debugging tool, an in-circuit emulator which in turn uses JTAG as the transport mechanism enables a programmer to access an on-chip debug module which is integrated into the CPU via JTAG. The debug module enables the programmer to debug the software of an embedded system.

A JTAG interface is a special four/five-pin interface added to a chip, designed so that multiple chips on a board can have their JTAG lines daisy-chained together, and a test probe need only connect to a single "JTAG port" to have access to all chips on a circuit board. The connection pines are:

- TDI (Test Data In)
- TDO (Test Data Out)
- TCK (Test Clock)
- TMS (Test Mode Select)
- TRST (Test ReSeT) optional.

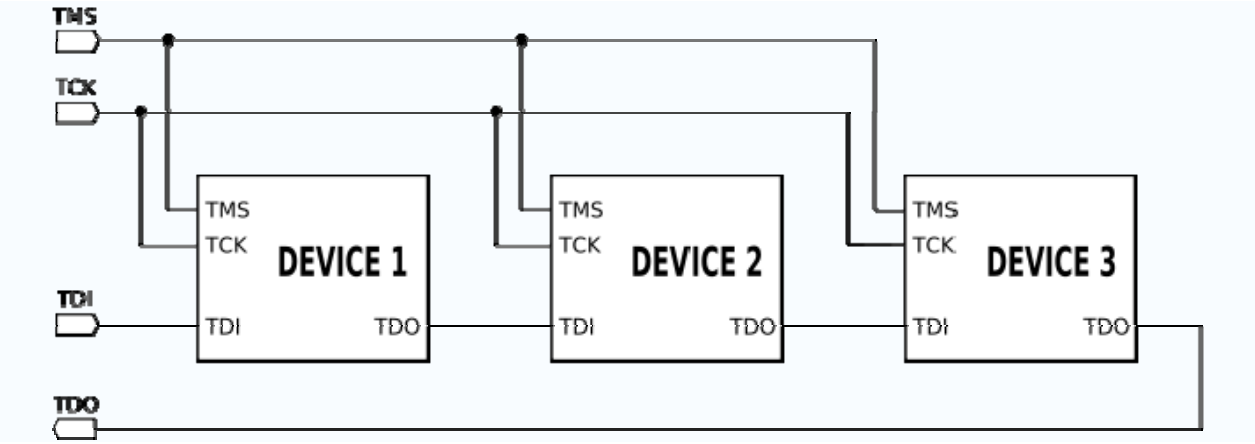


Figure 20 : JTAG topology

Since only one data line is available, the protocol is necessarily serial. The clock input is at the TCK pin. Configuration is performed by manipulating a state machine one bit at a time through a TMS pin. One bit of data is transferred in and out per TCK clock pulse at the TDI and TDO pins,

respectively. Different instruction modes can be loaded to read the chip ID, sample input pins, drive (or float) output pins, manipulate chip functions, or bypass (pipe TDI to TDO to logically shorten chain of multiple chips). The operating frequency of TCK varies depending on the chip.

When performing boundary scan on integrated circuits, the signals manipulated are between different functional blocks of the chip, rather than between different chips.

The TRST pin is an optional active-low reset to the test logic, usually asynchronous, but sometimes synchronous, depending on the chip. If the pin is not available, the test logic can be reset by clocking in a reset instruction synchronously.

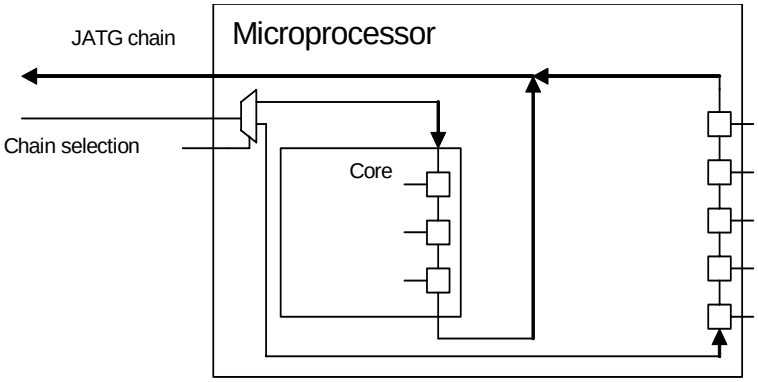


Figure 21 : JTAG chain

The principle used for the debug is the same one, chain JTAG not passing outside the circuit, but in the core, making it possible to read and write in all the registers. In fact, it is possible to select one of two JTAG chains in order to reach either the core or the pins.

8.3.2 Embedded peripheral

This section provides an overview of the advanced peripheral functions available on some microcontroller unit (MCU) in the AT91 series. It is important to emphasis the fact that ATMEL is not just another ARM baser microcontroller supplier. The core is one thing but the richness and complexity of associated peripheral will make whole architecture a real solution for embedded applications.

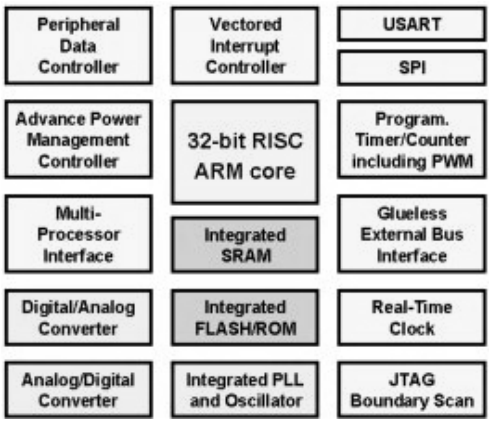


Figure 22 : AT91 embedded peripheral

The AT91 microcontrollers integrate several peripherals, which are classified as system or user peripherals. All on-chip peripheral are 32-bit accessible by the AMBA bridge and can be programmed with a minimum number of instructions. The peripheral register is composed of control, mode, data, status and enable/disable/status register.

Figure 23 : Peripheral register

8.3.3 Peripheral overview

- System peripheral
  - External Bus Interface (EBI)
  - Advanced Interrupt Controller (AIC)
  - Parallel I/O Controller
  - Watchdog
  - Peripheral Data Controller (PDC)
  - Advanced Power Management Controller (APMC)
  - Real Time Clock (RTC)
- User Peripherals
  - USART
  - Serial Peripheral Interface (SPI)
  - Timer Counter
  - Analog to Digital Converter (ADC)
  - Digital to Analog Converter (DAC)

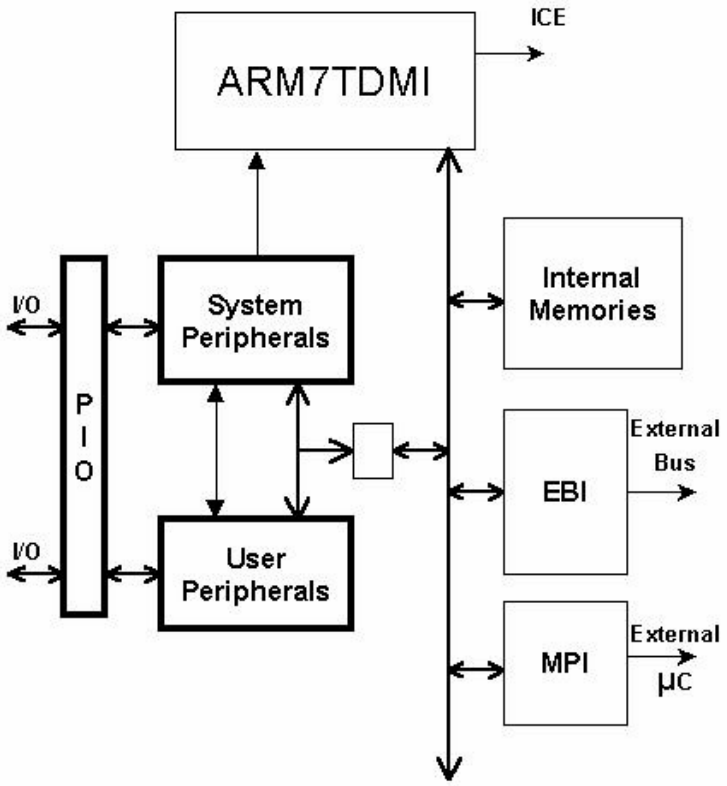


Figure 24 : Peripheral interconnection

8.3.4 External Bus Interface (EBI)

The EBI generates the signals that control the access to the external memory of peripheral devices.

The External Bus Interface provides those different features:

- Up to 8 programmable chip select lines
- Remap command which allows dynamic exception vectors
- Glue-less for both 8-bit and 16-bit standard memories
- 16-bit memories emulated with two 8-bit memories
- Up to 8 wait states can be programmed
- External wait request supported
- Early read protocol which allows faster clock with slower RAM
- Up to 7 Data Float Time can be programmed

This interface has 8 chip selects and a 24-bit address bus. The remap command enables switching between the boot memory and the internal RAM bank addresses allowing the exception vectors mapped from address 0x0 to 0x20 to be redefined dynamically by the software programs.

The 16-bit data bus can be configured to interface with 8-bit or 16-bit external devices. Separate read and write control signal allow for direct memory and peripheral interfacing and two 8-bit data-path memories can be emulated like a 16-bit data-path memory. Up to 8 standard Wait States can be software programmed to access slow peripherals. To insert more Wait States, an external Wait States

request may be presented with the assertion of the NWAIT signal. An early Read protocol has been defined to allow the use of faster clock with slower RAM. A Data Float Time (DFT) may be programmed to allow connection of devices that free too slowly the data-bus after a read access.

The following figure presents the block diagram of the EBI.

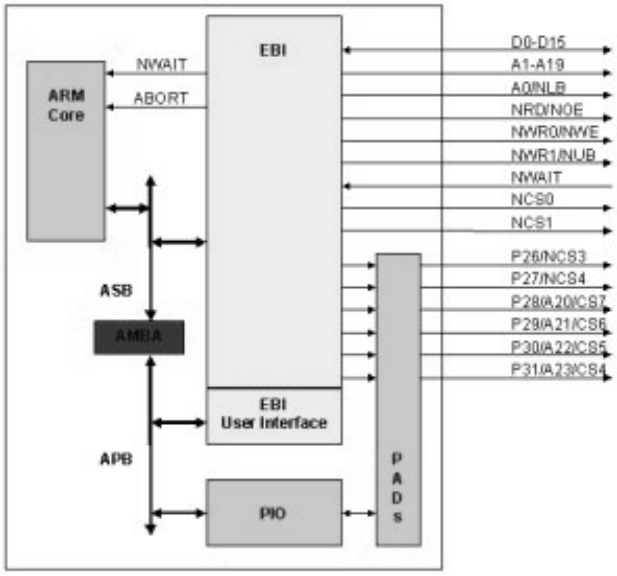


Figure 25 : External Bus Interface

Depending on data-bus path and their control signals, different possible connections with external devices are available. Each chip select with a 16-bit data-path bus can operate with one of two different types of write access.

- Byte select access type: one actual 16-bit memory

  Byte select access selects upper and/or lower byte with two byte select lines (NLB and NUB), and separate read and write signals (NRD and NEW). So, it is used to connect 16-bit devices in a memory page and if the device supports byte access, NLB enables the lower byte, NUB enables the upper byte.

- Byte write access type: two 8-bit memories

  Byte write access supports two byte write signals (NWR0 and NWR1) and a single read signal (NRD). So, it is used to connect two 8-bit devices as a 16-bit memory page. NWR1 enables upper byte writes and NWR0 enables lower byte writes.

Those two different connections possibilities are presented on the figure below.

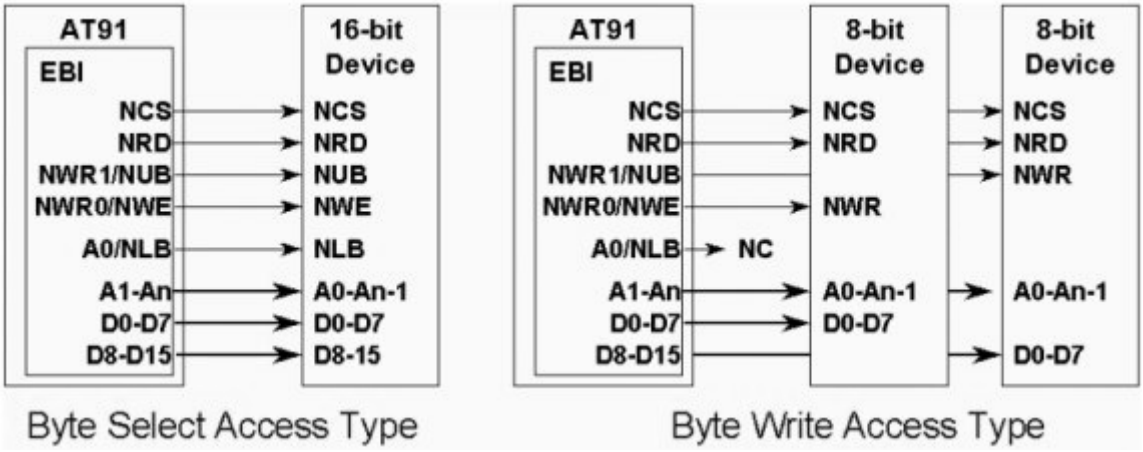


Figure 26 : Byte select / write access type

Read access may be handled with two different protocols to take into account peripheral particularities. The standard read protocol implements a read cycle in which NRD and NEW is similar. Both are active during the second half of the clock cycle. Early read protocol provides more time for a read access from the memory by asserting NRD at the beginning of the clock cycle. In the case of successive read cycles in the same memory, NRD remains active continuously. So, Early read protocol ca allow a faster clock frequency to be used or work faster by eliminating a Wait state.

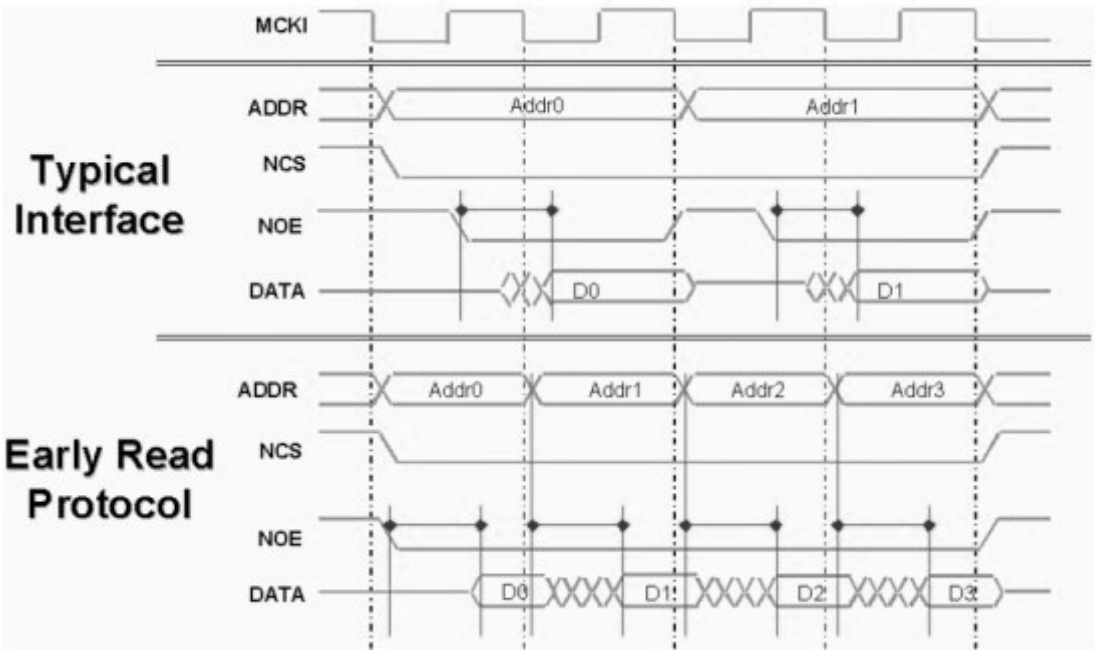


Figure 27 : Read access protocol

In Early read protocol, the EBI adds automatically one wait state after an external write access to remove any data bus contention risks.

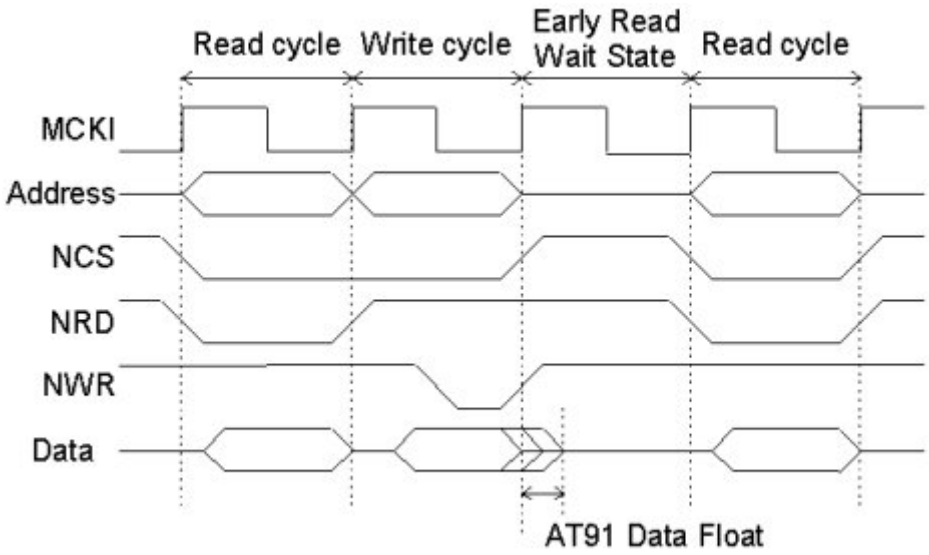


Figure 28 : Early read protocol

An early read wait state is automatically inserted when an external write cycle is followed by a read cycle to allow time for the write cycle to end before the subsequent read cycle begins. This wait state is generated in addition to any other programmed wait states. No wait state is added when a read cycle is followed by a write cycle, between consecutive accesses of the same type or between external and internal memory accesses. Early read wait states affect the external bus only. They do not affect internal bus timing.

Each chip select can be programmed to insert one or more (up to eight) wait states during an access on the corresponding device. Note that if NRD is always de-asserted on the MCKI rising edge of the last access cycle, it is not the case for NWR. If standard wait states are added, NWR is de-asserted one half cycles before the end of the access. A chip select wait state is added when consecutive accesses are made to two different external memories. It is not added if the first access is done with standard wait states.

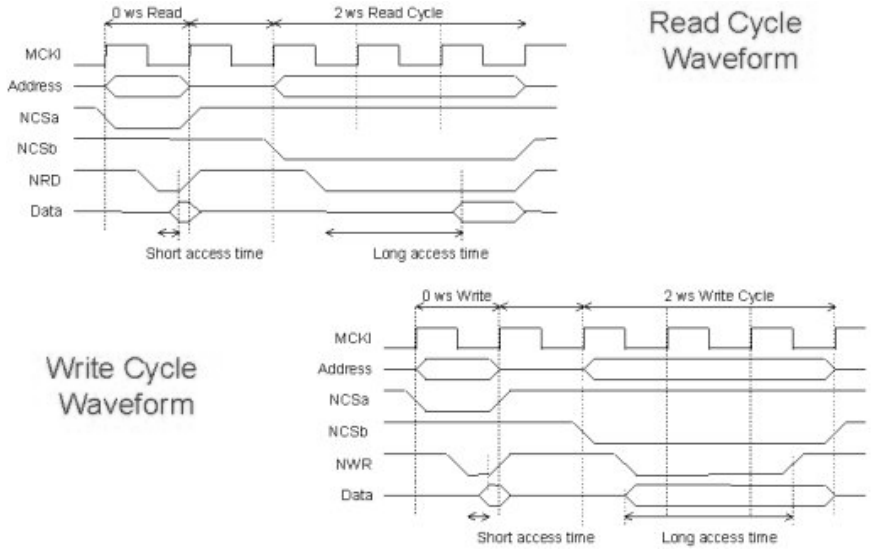


Figure 29 : Read / Write cycle waveform

The next two paragraphs present an example of wait state calculation.

- Wait state calculation: Read access

  The parameters to be considered are the minimum time required after chip enable (tCE) and output enable (tOE) are detected low to put valid data on the bus. The datasheet of the AT49BV1604-90 give those different values: 90[ns] max for tCE and 40[ns] max for tOE.

  With those values for a standard read the requirements are:

$$\begin{aligned} ntCP - EBI4 - EBI25 &\geq tCE \\ ntCP - \frac{tCP}{2} - EBI22 - EBI25 &\geq tOE \end{aligned}$$

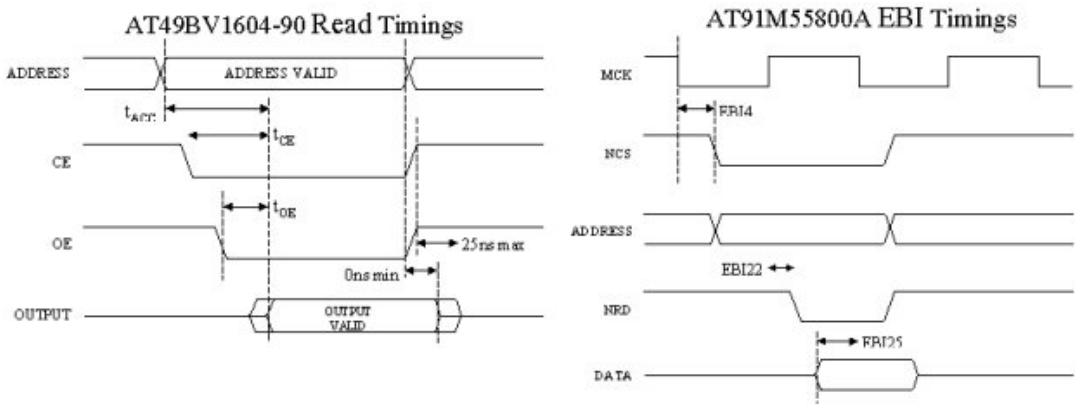


Figure 30 : Read access wait state

- Wait state calculation: Write access

  For a write access the parameters to be considered are tWP which is 100[ns] min and tDS which is 100[ns] min.

  The requirements are, with n is the number of clock cycle:

$$\begin{aligned} (n-1)tCP - EBI8 + EBI10 &\geq tWP \\ (n-1)tCP - EBI11 + EBI10 &\geq tDS \end{aligned}$$

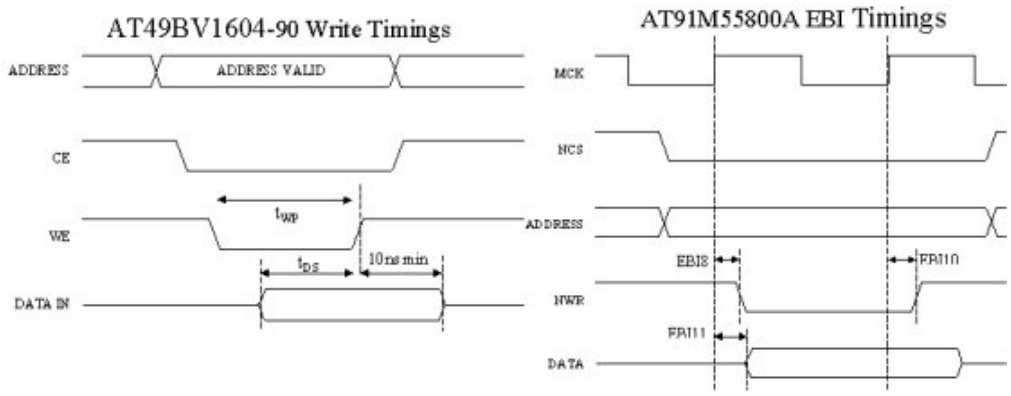


Figure 31 : Write access wait state

Some memory devices are slow to release the external bus. For such devices it is necessary to add wait states (data float waits) after a read access before starting a write access or a read access to a different external memory. It represents the time allowed for the data output to go high impedance after the memory is disabled. The following figure presents 2 different devices which are slow releasing the data bus. Read device 'a' with two tdf cycle and read device 'b' with three tdf cycles.

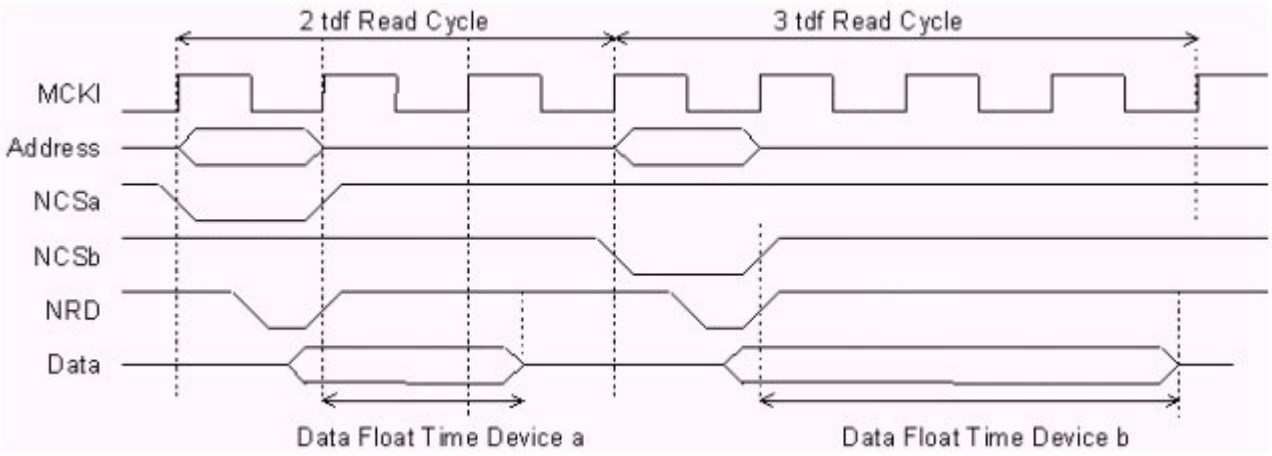


Figure 32 : Data float time

The ARM vectors are mapped from address 0x0 to address 0x20. The remap command allows these vectors to be redefined dynamically by the software. The remap consists in switching between the boot memory and the internal RAM bank addresses.

Before the remap command, the only peripherals accessible are the internal memories and registers and the boot memory, selected by the chip select 0. The user can modify the chip select 0 configuration, programming the EBI_CSR0 with the exact boot memory characteristic.

- Default Memory 0 configuration:
  - CSR0 = 0x0000203D or 0x0000203E
    - Eight wait states
    - 0 data float time
    - 8/16 bits data bus width selected by BMS

The base address becomes effective after the remap command, but the new number of wait states can be changed immediately. This is useful if a boot sequence needs to be faster.

- Speed up of the boot sequence before remap
  - Can be performed by writing EBI_CSR0

The base addresses are defined in the EBI registers, in EBI_CSR0 register for the memory connected to CS0. This base address generally corresponds to the link address.

- Base addresses are defined in EBI registers
  - In EBI_CSR0 for memory 0
  - Generally corresponds to the link address

The figure below shows the difference of memory mapping before and after remap command.

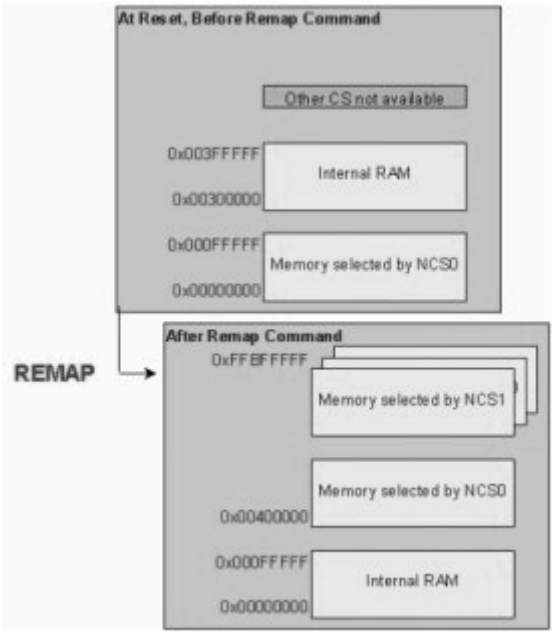


Figure 33 : Remapping

8.3.5 Advanced Interrupt Controller (AIC)

The AT91 microcontroller embeds an 8-level priority, individually maskable, vectored interrupt controller. This feature substantially reduces the software and real time overhead in handling internal and external interrupts. Internal sources are programmed to be level sensitive or edge triggered. External sources can be programmed to be positive or negative edge triggered or high or low level sensitive.

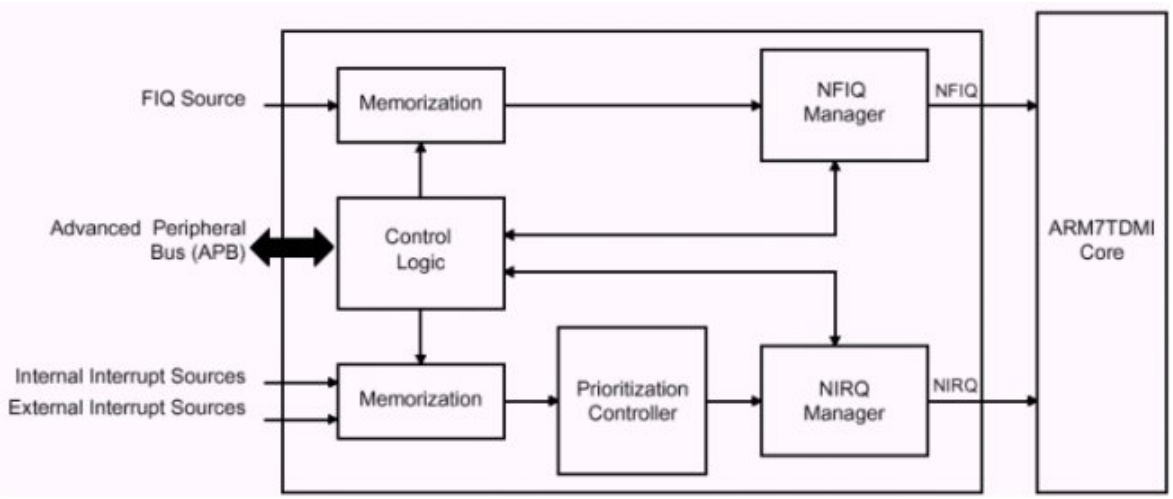


Figure 34 : AIC block diagram

8.3.6 Parallel Input/Output Controller (PIO controller)

The AT91M55800A microcontroller has two PIO Controllers. The PIO Controller has 32 programmable IO lines, with pins dedicated as general purpose IO pins and the other IO lines are multiplexed with an external signal of a peripheral to optimize the use of available package pins.

The PIO Controller enables the generation of an interrupt on input change on any of the PIO pins. After Reset, the pin is controlled by the PIO Controller and is in input.

Each IO can be programmed for multi-driver option. This means that the IO is configures as open drain in order to support external drivers on the same pin. An external pull-up is necessary to guarantee a logic level of one when the pin is not being driven.

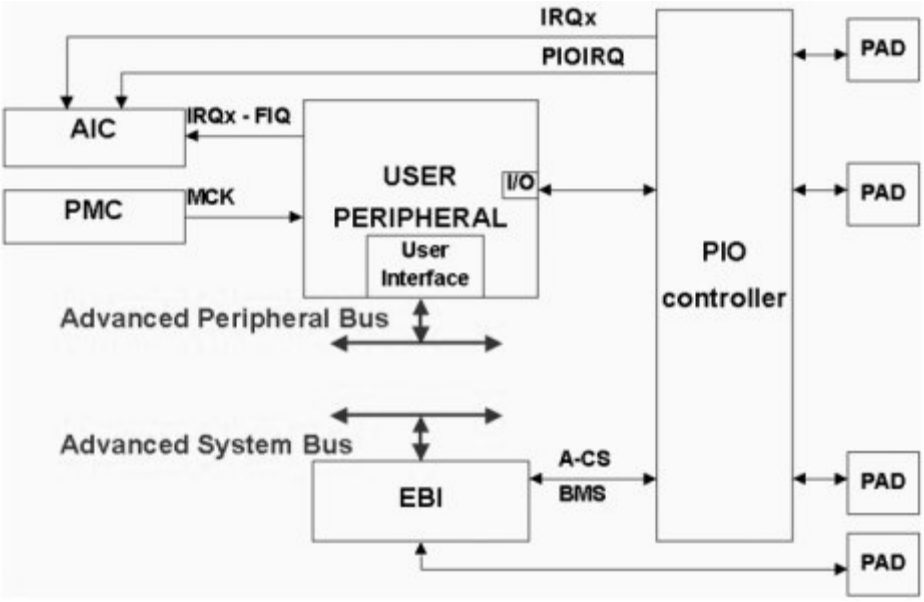


Figure 35 : PIO controller block diagram

Each pin can be configured to be driven high or low. The level is defined in four different ways, according to the following conditions:

- If a pin is controlled by the PIO Controller and is defined as an output, the level is determined by the external circuit.
- If a pin is controlled by the PIO Controller and is defined as an output, the level is programmed using the registers Set Output Data (PIO_SODR) and Clear Output Data (PIO_CODR).
- If a pin not controlled by the PIO Controller, the state of the pin is defined by the peripheral.
- In all cases, the level on the pin can be read in the register PIO_PDSR (Pin Data Status).

8.3.7 Watchdog

The watchdog has a 16-bit down counter. Bits 0 to 11 set (FFF) and bits 12 to 15 are user definable. Four clock sources are available to the watchdog counter (MCK32, MCK/128, MCK/1024 and MCK/4096) and provide a programmable time-out period of 4[ms] to 8[s] with a 33[MHz] system clock. If an overflow occurs, the watchdog can generate three independent outputs; an internal reset, internal interrupt or a low level on the NWDOF signal for duration of eight MCK cycles. All write accesses are protected by control access keys to help prevent corruption of the watchdog.

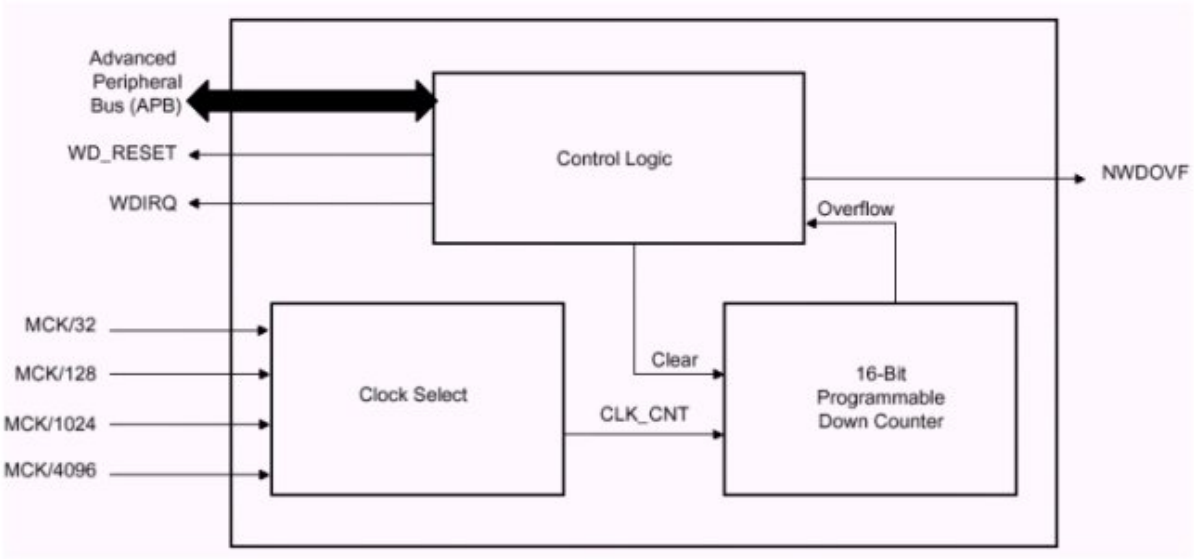


Figure 36 : Watchdog block diagram

The watchdog timer can be used to prevent system lock-up if the software becomes trapped in a deadlock. In normal operation the user reloads the watchdog at regular intervals before the timer overflow occurs. If an overflow does occur, the watchdog timer generates one or a combination of signals.

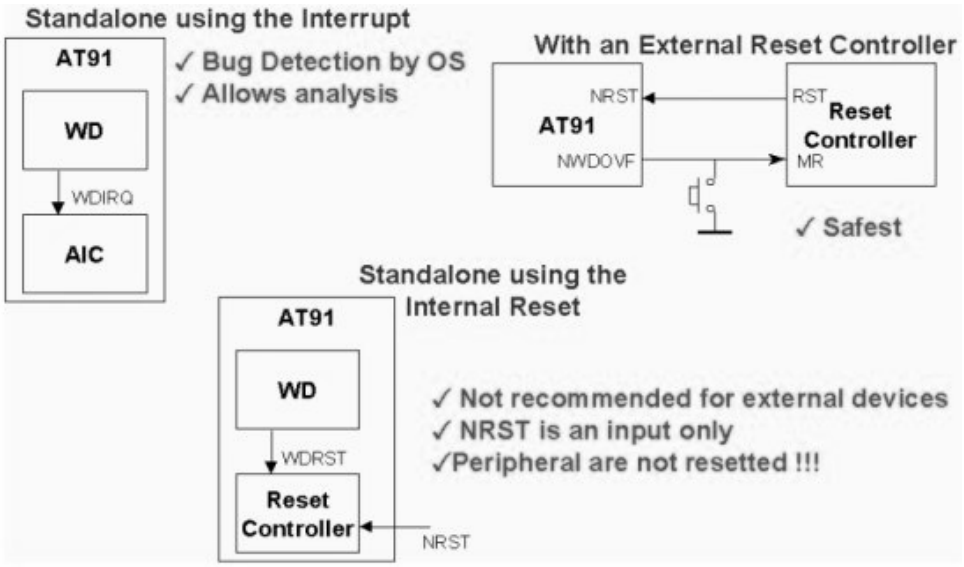


Figure 37 : Software checking solutions

8.3.8 Advanced Power Management Controller (APMC)

The ARM7TDMI processor is industry leader in MIPS/Watt. The AT91 microcontroller embeds power management controllers, which provide idle mode and disable clock on unused peripherals.

An Advanced Power Management Controller (APMC) is available on the AT91M55800A microcontroller. The APMC optimizes both the power consumption of the device and the complete system. The APMC controls the clocking elements such as the oscillator and the PLL, core and the peripherals clock. Moreover it has the capability to control the system power supply.

Five operating modes are supported by the APMC and offer different power consumption levels and event response latency times:

- Normal mode
  - The AR core clock is enabled
  - Used peripheral clocks are enabled
- Idle mode
  - The ARM core clock is disabled and waiting for the next interrupt
  - The peripheral clocks are enabled and the PDC transfers are still possible
- Slow clock mode
  - The device core and peripheral run in slow clock mode
- Standby mode
  - Combination of slow clock and idle mode which provides very low power consumption
- Power down mode
  - The main power supply is turned off until a programmable edge on the wake-up signal or a programmable RTC alarm occurs

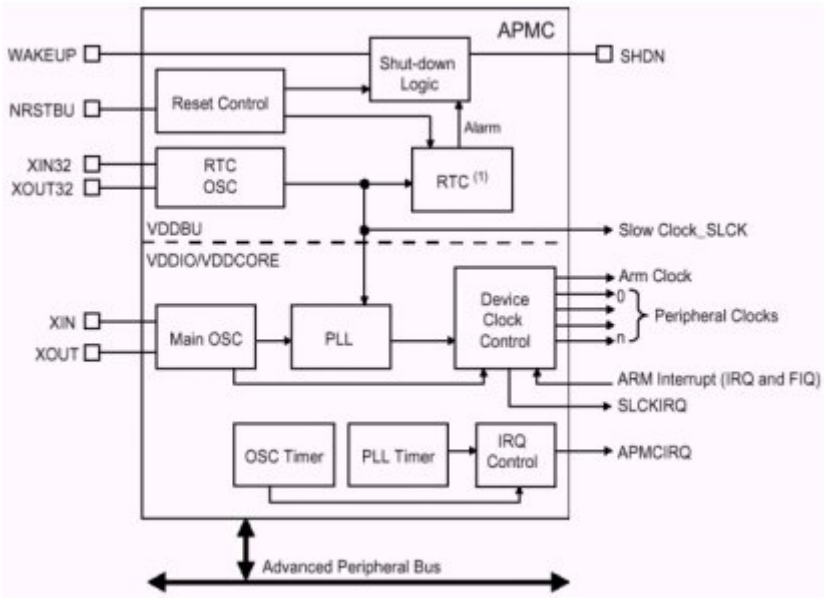


Figure 38 : APMC block diagram

The APMC consists of the following elements: The RTC Oscillator which provides the slow clock at 32768Hz, the main oscillator, the phase lock loop, the ARM processor clock controller which allows entry to the idle mode, the peripherals clock controller which conserves the power consumption of unused peripherals, the master clock output controller and the shut-down logic which controls the main power.

8.3.9 Real Time Clock (RTC)

The AT91M55800A features a Real Time Clock (RTC) peripheral that is designed for very low power consumption. It combines a complete time-of-day clock with alarm and a two hundred year calendar, completed by a programmable periodic interrupt. The time and calendar value are coded in Binary Coded Decimal (BCD) format. The RTC provides five programmable fields: Month, Date, Sec, Min and Hour.

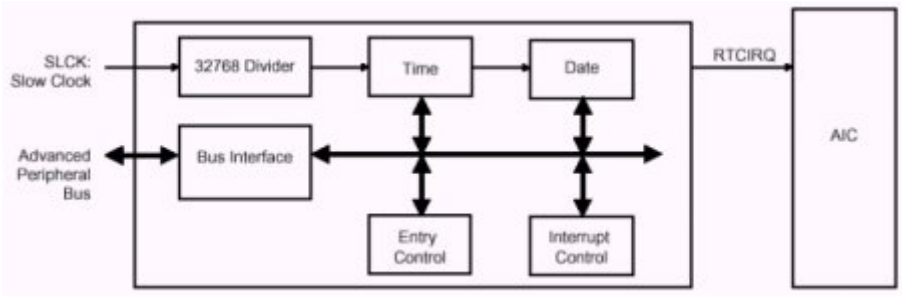


Figure 39 : RTC block diagram

8.3.10 USART

The AT91M55800A has three identical, full duplex, universal synchronous/asynchronous receiver/transmitters which are connected to the Peripheral Data Controller.

The USART features:

- Programmable baud rate generator with external or internal clock
  - Up to 1 [Mbits/s] in Asynchronous mode and up to 16 [Mbits/s] in synchronous mode at 32[MHz]
- Parity, Framing and Overrun error detection
- Line break generation and detection
- Automatic echo, local loop back and loop back channels modes
- Multi drop mode: address detection and generation
- Interrupt generation
- Two dedicated PDC channels
- 5, 6, 7, 8 and 9-bit character length

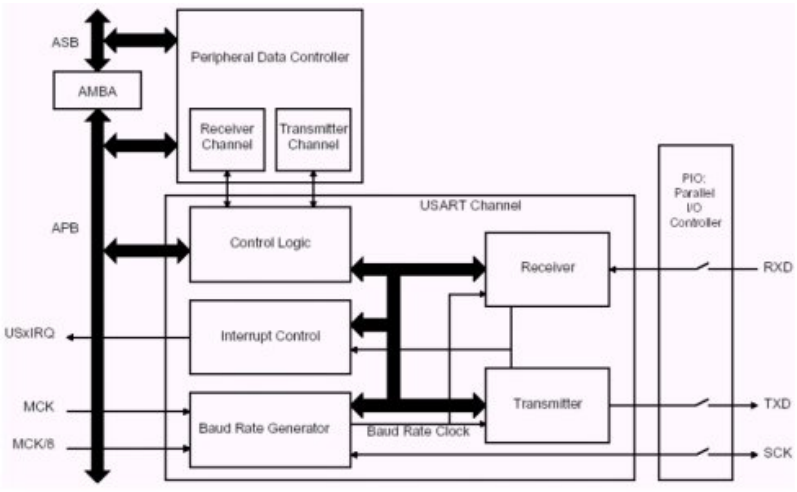


Figure 40 : USART block diagram

The Baud rate generator provides the bit period clock to both the receiver and the transmitter. The baud rate generator can select between external and internal clock sources. The external clock source is SCK and the internal clock sources can be either the master clock MCK or the master clock divided by eight (MCK/8). In asynchronous mode, the baud rate equals the selected clock divided by 16 times CD. For synchronous mode: Baud rate = Selected clock/CD.

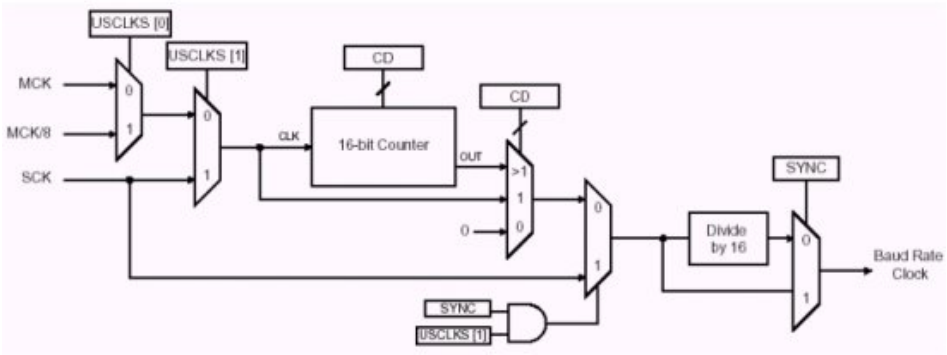


Figure 41 : USART baud rate generator

- Reception

  In asynchronous mode, the USART detects the start of a received character by sampling the RXD signal until it detects a valid start bit. When a valid start bit has been detected, the receiver samples the RXD at the theoretical mid-point of each bit. In synchronous mode, the receiver samples the RXD signal on each rising edge of the baud rate clock.

  - Asynchronous: 8 bit, 1 start and 1 stop

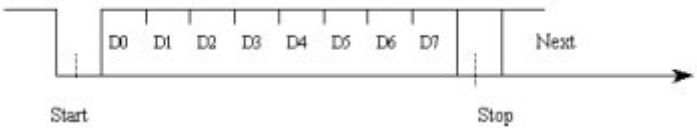


Figure 42 Asynchronous reception

- Synchronous: 8 bit, 1 start and 1 stop

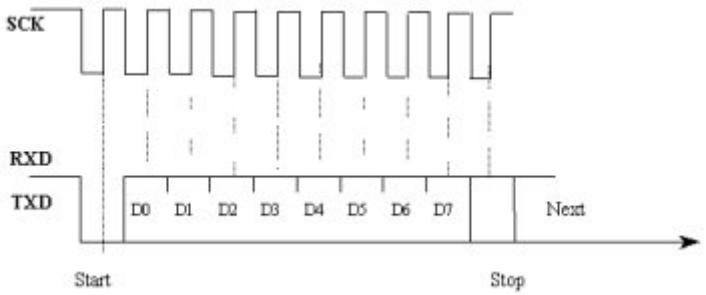


Figure 43 : Synchronous reception

- Transmission

  The transmitter has the same behavior in both synchronous and asynchronous operating modes. Start bit, data bits, parity bit and stop bits are serially shifted, lowest significant bit first, on the falling edge of the serial clock.

  - Asynchronous and synchronous: 8 bit, 1 parity and 1 stop

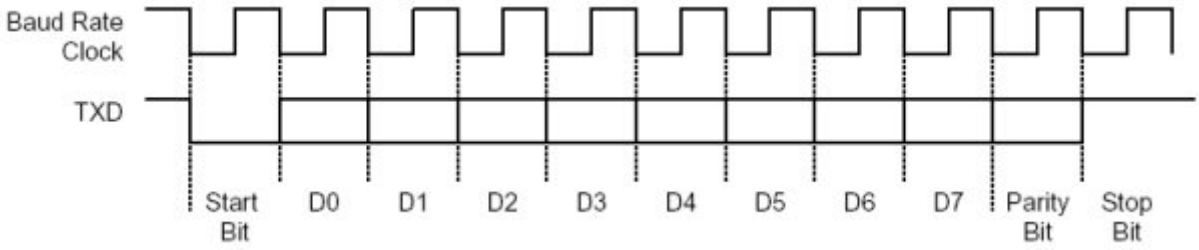


Figure 44 : Asynchronous and synchronous transmission

The PDC channels allow transferring data from on-chip peripheral such as USART, SPI to on-chip memories without CPU intervention. Each USART channel is closely connected to a corresponding Peripheral Data Controller channel. One is dedicated to the receiver; the other is dedicated to the transmitter.

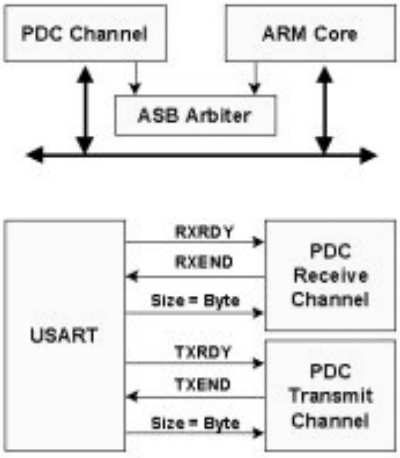


Figure 45 : PDC Channel

- PDC shares the ASB bus with the ARM core
  - External or internal memories access
  - ARM core stopped during 3 cycles minimum.
- Each PDC channel is dedicated to a peripheral and a transfer direction
  - PDC registers mapped in User interface
  - End of transfer in the Status register
- Typical application
  - Code download
  - Packet exchange
    - Receiver timeout helps to support variable length packets
    - Transmitter tie guard helps to support slow remote devices

8.3.11 Serial Peripheral Interface (SPI)

The AT91M55800A has one SPI which provide communication with external devices in master or slave mode. The SPI has four external chip selects which can be connected to up to 15 devices.

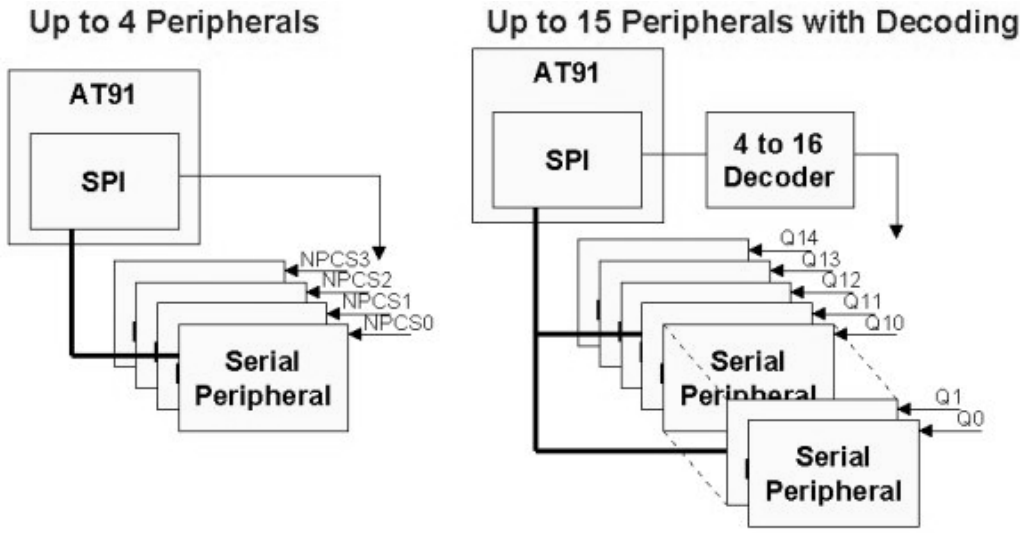


Figure 46 : Bus Implementation

The data length is programmable from 8 to 16-bit. As for the USART, a 2-channel PDC can be used to move data between memory and the SPI without CPU intervention. The SPI features a full duplex three wires synchronous transfer, the MISO: Master In Slave Out, MOSI: Master Out Slave In; SPCK: SPI Clock. The maximum SPI baud rate clock is MCK/4 and provides Fault Detection in master mode.

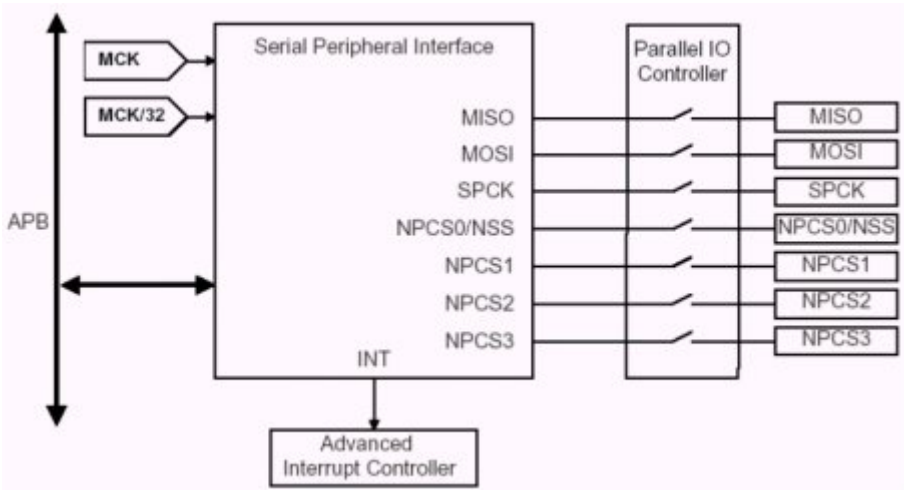


Figure 47 : SPI block diagram

8.3.12 Timer Counter

The AT91M55800A features two Timer Counter blocks, each containing three identical 16-bit counter channels. Each channel can be independently programmed to perform a wide range of functions including frequency measurement, event counting, interval measurement, pulse generation, delay timing and pulse-width modulation.

Each timer Counter channel has three external clock inputs, five internal clock inputs, and two multipurpose input/output signals which can be configured by the user. Each channel drives an internal interrupt source which can be programmed to generate processor interrupts via the AIC.

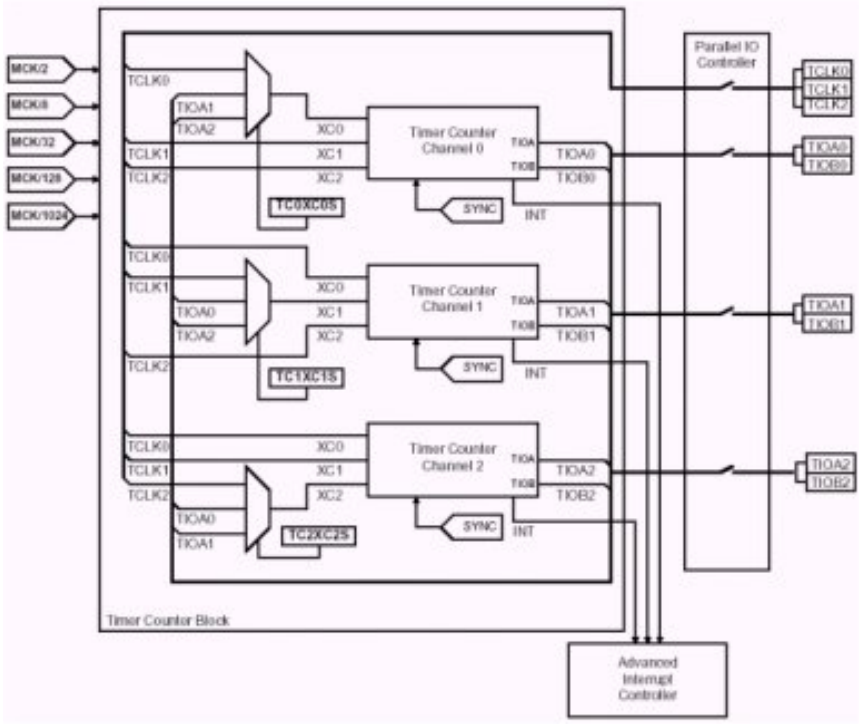


Figure 48 : Timer Counter block diagram

Each channel can independently select an internal or external clock source for its counter. Internal clock signals: MCK/2, MCK/8, MCK/32, MCK/128 and MCK/1024. External clock signals: XC0, XC1 and XC2. The selected clock can be inverted to allow counting on the opposite edges of the clock. The burst function allows the clock to be validated when an external signal is high.

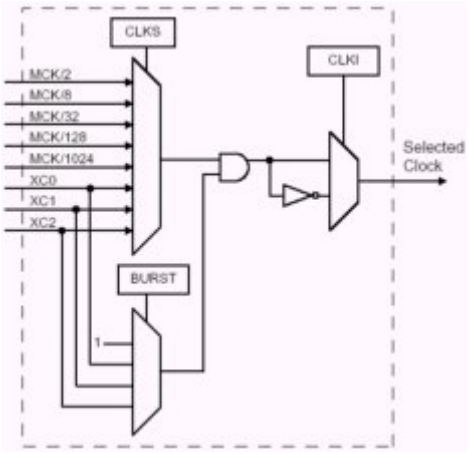


Figure 49 : Clock selection

The clock of each counter can be controlled in two different ways; it can be enabled/disabled by control register CLKEN and CLKDIS and started/stopped. Loading RB (capture register B) in capture mode or RC (compare register C) in Waveform mode can stop or disable the counter clock.

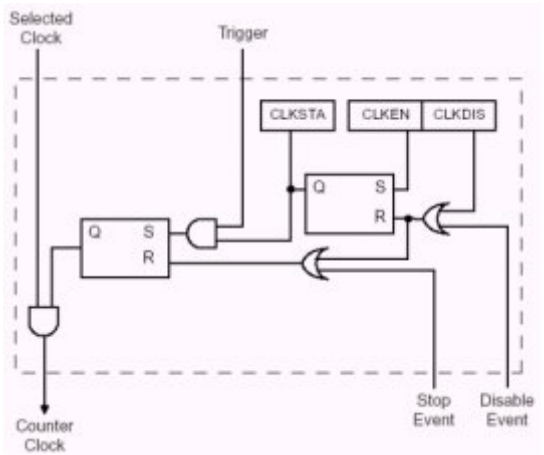


Figure 50 : Clock control

Each timer counter channel can independently operate in two different modes: Capture mode which allows measurement on signals and waveform mode which allows wave generation. The different timer counter mode can be programmed with the WAVE bit in the TC mode register.

A trigger resets the counter and starts the counter clock. The three following triggers are common to both modes:

- Software Trigger

  Each Channel has a software trigger, available by setting SWTRG in TC_CCR

- SYNC

  Each channel has a synchronization signal SYNC. When asserted, this signal has the same effect as a software trigger. The SYNC signals of all channels are asserted simultaneously by writing TC_BCR (block control) with SYNC set.

- Compare RC Trigger

  RC is implemented in each channel and can provide a trigger when the counter value matches the RC value if CPCTRG is set in TC_CMR.

In capture mode TIOA and TIOB, in waveform mode TIOB, XC0, XC1, XC2 can be used as external trigger.

Capture mode allows the TC channel to perform measurements such as pulse timing, frequency, period, duty cycle and phase on TIOA and TIOB signals which are considered as input. Register A and register B are used as capture registers. This means that they can be loaded with the counter value when a programmable event occurs on signals TIOA and TIOB.

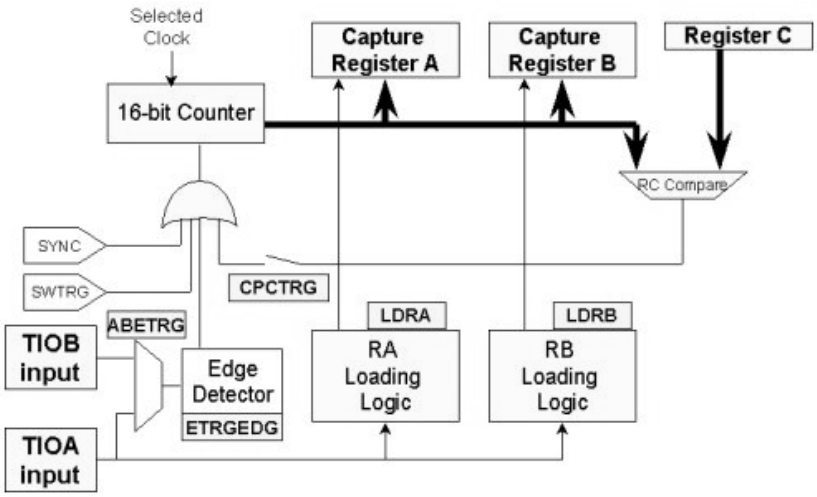


Figure 51 : Capture mode

- Measure the phase between TIOB and TIOA and the duration of the TIOA pulse
  - TIOB rising edge resets and starts the counter
  - TIOA rising edge loads RA and a falling edge loads RB

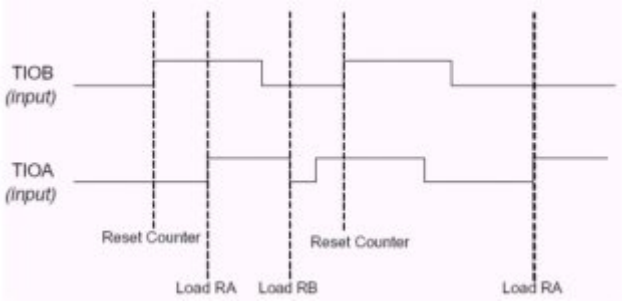


Figure 52 : Phase measurement

  - RA contains the phase between TIOB and TIOA
  - (RB-RA) is the duration of the TIOA pulse
- Measure the duration of a TIOA pulse or period
  - TIOA falling edge resets and starts the counter and loads RB if RA is already loaded
  - TIOA rising edge loads RA

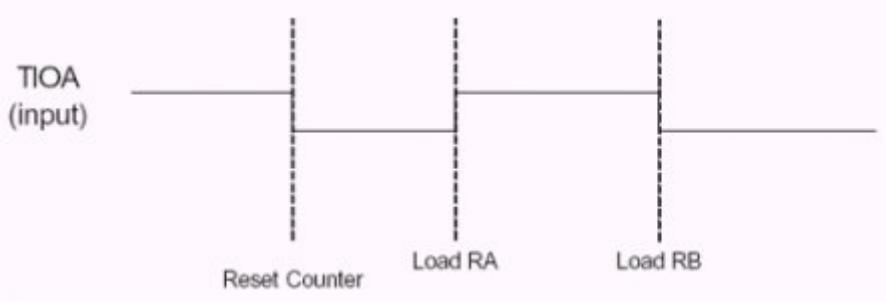


Figure 53 : Duration measurement

  - RA contains the duration of a TIOA pulse (low level)
  - RB contains the duration of the TIOA period

Waveform mode allows the TC channel to generate one or two PWM signals with the same frequency and independently programmable duty cycles, or to generate different types of one-shot or repetitive pulses. In this mode, TIOA is configured as output and TIOB is defined as output if it is not used as an external event. RA, RB and RC are all used as compare registers.

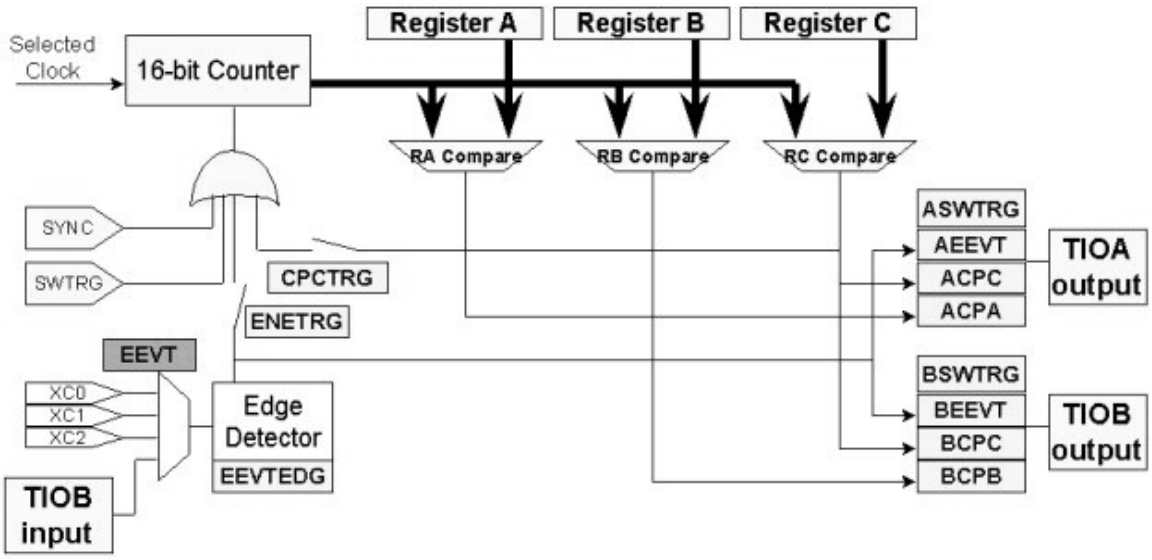


Figure 54 : Waveform mode

- Dual Pulse Width Modulation (PWM) generation
  - TIOA is toggled by RA and RC, TIOB by RB and RC
  - A trigger starts the counter and initializes TIOA and TIOB

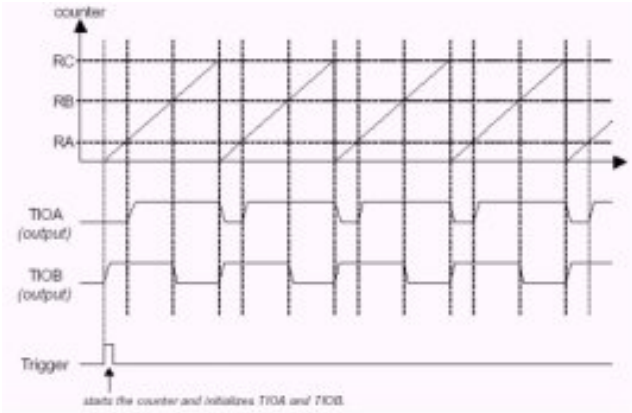


Figure 55 : PWM in waveform mode

  - The PWM frequency must be stored in the compare register RC
  - The duty cycles on TIOA and TIOB are defined by RA and RB respectively

8.3.13 Analog to Digital Converter (ADC)

The AT91M55800A features two identical four channel 10-bit Analog-to-digital converters (ADC) based on a Successive Approximation Register (SAR) approach.

Each ADC has four analog inputs pins (AD0 to AD3 and AD4 to AD7), digital trigger inputs pins (AD0TRIG and AD1TRIG), and provides an interrupt signal to the AIC. Both ADC share the analog power supply pins (VDDA and GNDA) and the input reference voltage pin (ADVREF).

The ADC features:

- Two identical four channel ADC
  - 10-bit resolution
  - Successive Approximation Register (SAR) approach
  - Settable analog input conversion range (dedicated VREF)
  - 11 ADC clock cycles conversion time including 1 ADC clock cycle for sample and hold
  - Four LSB maximum integral non-linearity
  - Sleep mode (energy saving)
- Starting modes:
  - Software trigger, External input (A/D trigger), Timers on-chip event signal
- Dedicated analog power supply pins (VDDA and GNDA)
- End of conversion interrupt

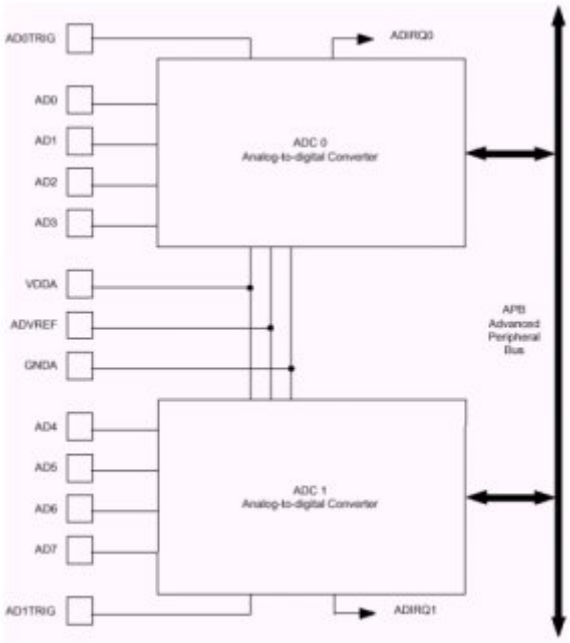


Figure 56 : ADC block diagram

8.3.14 Digital to Analog Converter (DAC)

The AT91M55800A features two identical one channel Digital to Analog Converters (DAC).

Each DAC has an analog output pin (DA0 and DA1) and provides an interrupt signal to the AIC (DA0IRQ and DA1IRQ).

The DAC features:

- Two identical one channel DAC
  - 10-bit resolution
  - 6[µs] maximum setting time
  - Settable analog output range (dedicated VREF)
  - Four LSB maximum integral non-linearity
- Starting mode
  - Software trigger
  - Timers on-chip event signal
- Dedicated analog power supply pins (VDDA and GNDA)
  - Improve noise rejection
- Data ready interrupt

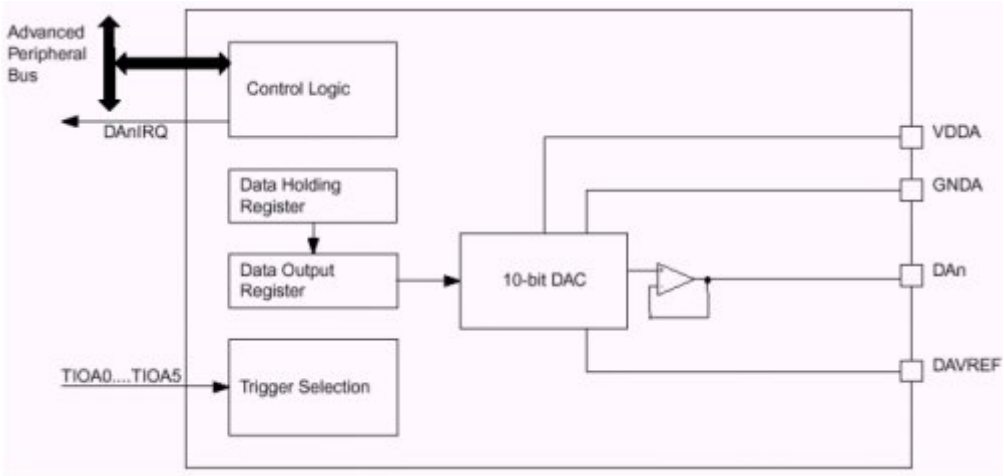


Figure 57 : DAC block diagram

8.4 Preliminary Budget

This section will present a preliminary budget for the CDMS board. Only the main elements are presented and should provide a first conclusion about what the board will consume and weight.

The budgets were calculated with the component selected further. The following values are provided for an informative use only, and will not represent the final value.

- Mass budget

  Information on the masses of the components is usually not written on the datasheet. In the case of the weight is not indicated, the approximation will be done on the base of the weight of the ATMEL AT91M55800A microcontroller. This microcontroller was weighed and has typically a weight of 2[g] for a size of 26*26*1.6[mm]. The different components must be welt on a printed circuit board (PCB), which has a typical weight of 30[g] for an 80*80[mm] board.

- Size budget

  The size presented in the different table is for a typical thin quad flatpack (TQFP) package.

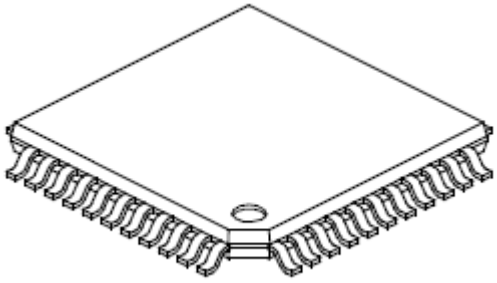


Figure 58 : TQFP package

- Power budget

The current consumption is the typical value presented on the different datasheet.

The components are presented on the following table. It is quite difficult to estimate the energy used by the system as those values are extracted from the different datasheet and will not always represent the final consumption. Moreover as the software is not done at that time, it is impossible to predict the activity duty cycle of the components.

Type	Manufacturer	Item	Package	Quantity	Lenght[mm]	Width[mm]	Height [mm]	Weight[mg]	Activity duty cycle	Average power [mW@1Mhz]
uC	ATMEL	AT91M55800A-33AU	LQFP 176	1	26	26	1.6	1800	1	10
Flash 16Mb	Spansion	S29AL016D-90-TFI-02-0	TSOP 48	2	12	20	1.2	600	0.5 0.5	5 10
Sram 4Mb	Renesas	R1LV0416CSB-7LI	TSOP 44	1	12	18	1.2	300	1	6
Prom 4Mb	ATMEL	AT27BV4096-12VI	TSOP 40	1	10	14	1.2	200	1	4
Silicon Osc.	MAXIM	MAX7375AXR405-T	SC70	1	2	2.2	1	40	1	10
Spi to IIC	Philips	SC18IS601IPW	TSOP 16	1	5	6.6	1.1	80	1	2

The size, weight and energy used for a frequency of 4[MHz] are calculated and are presented individually on the table below. On the right table, an estimation of the weight of the CDMS board has been made. This estimation takes into account the weight of components, PCB, connectors,

soldering and coating. This estimation will probably differ from the manufactured board but gives a good idea of what we have to expect.

item	size [mm^2]	weight [mg]	P [mW]	E [mWh]
uC	676	1800	40	40
Flash	480	1200	20 | 40	30
Sram	216	300	24	24
Rom	140	200	16	16
Osc.	4.4	40	10	10
Bridge	33	80	8	8

Item	Weight	Spec
Components	3580	30000
PCB	30000	30000
Connectors	25000	25000
Soldering	5000	
Coating	5000	
Total	68580	94000

This table compares the entire CDMS board budget with the specification. It is possible to see that the place available is quite small, it is required to place components on the two face of the PCB. The other values will respect the specification and it is now possible to continue with the hardware design of the board.

	Approx.	Spec	Comments
Weight [mg]	68580	94000	All CDMS board
Size[mm^2]	4635	3600*2	Coeff. 3 for routing and additional elements
P read[mW]	108	-	Value when all active in read mode
P write[mW]	128	-	Value when all active in write mode
E [mWh]	118	150	Approximated Energie used

Those values are only a preliminary budget and must be validated with the prototype board.

8.5 Hardware architecture

At that time the main components of the CDMS board have been selected, specifications are respected, before continuing the development with the schematic; hardware architecture has been drawn and is presented with the following figure.

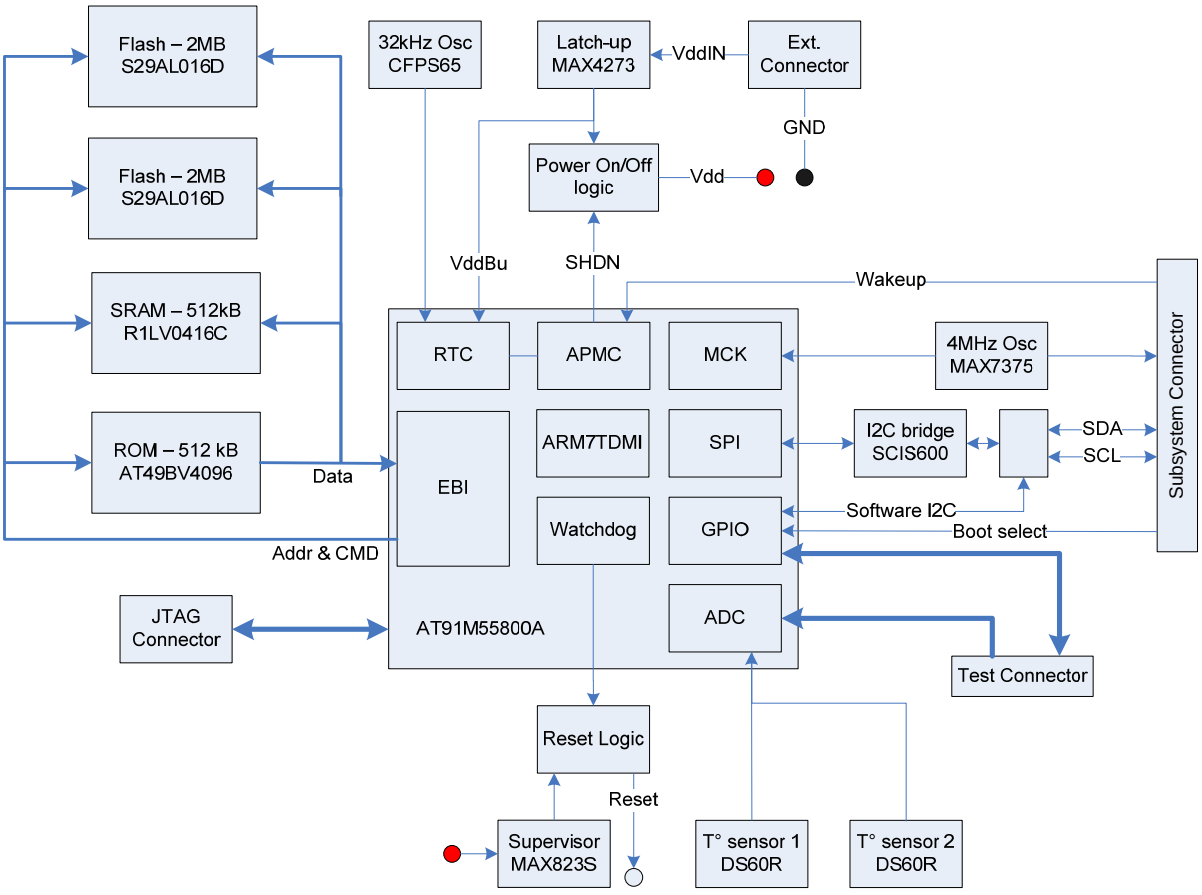


Figure 59 : Hardware architecture

8.5.1 Memory Mapping

The memory map allows determining the relationship between the logical addressing seen from the programmer and the material being physically present. The following figure presents the remap command. After a reset it is normal to find the external non-volatile memory starting from the address 0x0000, the core beginning the execution at this address.

Before Remap

0x00000000	External memory on NCS0
0x00100000	Reserved
0x00300000	Internal RAM
0x00400000	Reserved
0xFFC00000 0xFFFFFFFF	Internal peripherals

After Remap

0x00000000	Internal RAM
0x00100000	Reserved
0x00400000	External Peripherals (NCS0 -> NCS7)
0xFFC00000 0xFFFFFFFF	Internal peripherals

Figure 60 : Remapping

The following figure shows the memory mapping of the external memory for the CDMS. Four chip select are used. NCS0 is used for the external memory which will contain the flight software, NCS1 for the "Command" storage, NCS2 for the "Science" storage and NCS3 for the external SRAM memory chip.

0x00400000	Unused
0x01000000	External Flash or ROM on NCS0
0x01200000	Unused
0x02000000	External Flash on NCS1
0x02200000	Unused
0x03000000	External Flash on NCS2
0x03200000	Unused
0x04000000	External SRAM on NCS3
0x04080000	Free for NCS4 to NCS7
0xFFC00000 0xFFFFFFFF	Internal peripherals

Figure 61 : CDMS memory mapping

8.6 Schematic

The schematic has been made using P-CAD 2004 from Altium.

8.6.1 Alimentation

As presented in section 6.3.8, the AT91M55800A provide the feature to shut the main power down and power only the RTC and APMC with a back-up alimentation. This feature offer a shut-down mode which consume less than 10[µ W].

The following figure presents the implementation of this functionality.

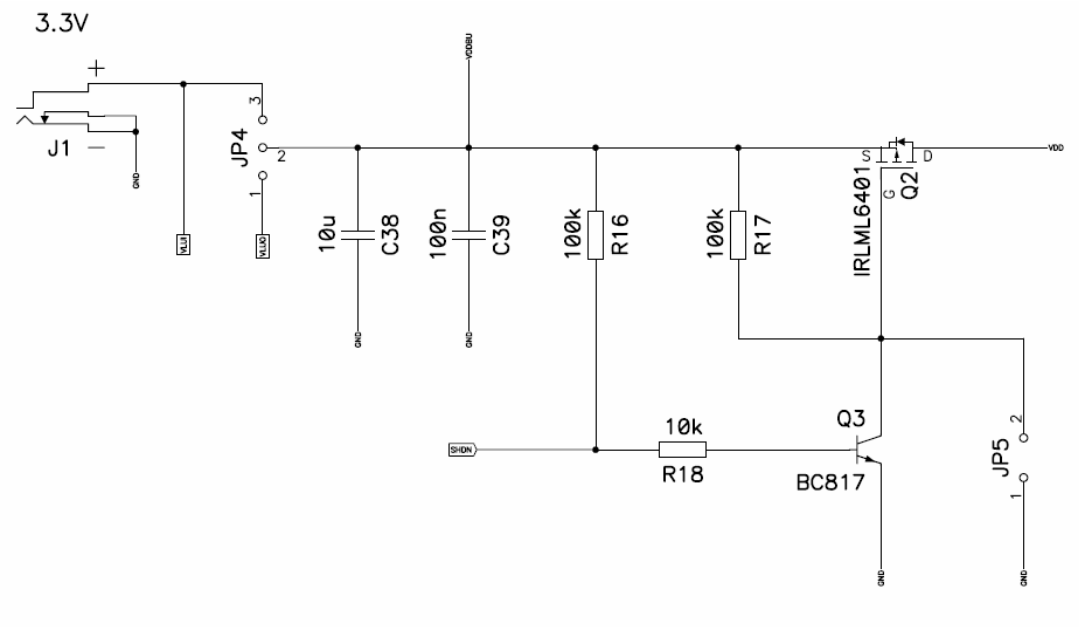


Figure 62 : Back-up alimentation

- VDDBU is the alimentation of the APMC and RTC.
- VDD is the alimentation of the whole board.
- VLUI and VLUO are the connection to the latch-up protection. JP4 let the possibility to use or not the latch-up protection functionality.
- J1 is the external connector for the alimentation entry. As the connector of the final board has not been yet decided, it is a standard low voltage connector which will be used only on the prototype board.
- SHDN is the command coming from the microcontroller; this line is pulled up by R16 and command the NPN transistor Q3 which will command the gate of the power MOSFET Q2. Q2 is able to dissipate more than 1[W] which should be sufficient regarding to the power consumption of the card.
- JP5 provide the ability to enable the shutdown feature if the jumper is open, or to disable it if closed.

A simulation has been made with ORCAD 10.5; the results are presented on following figures.

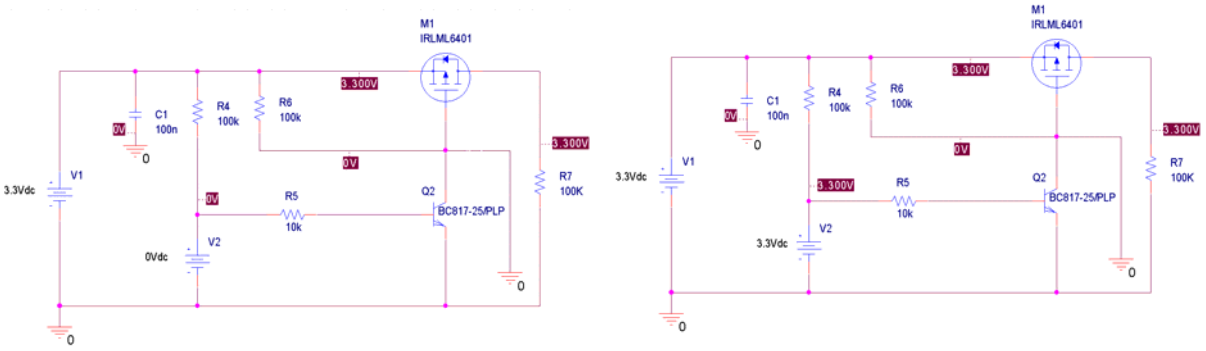


Figure 63 : Jumper JP5 closed

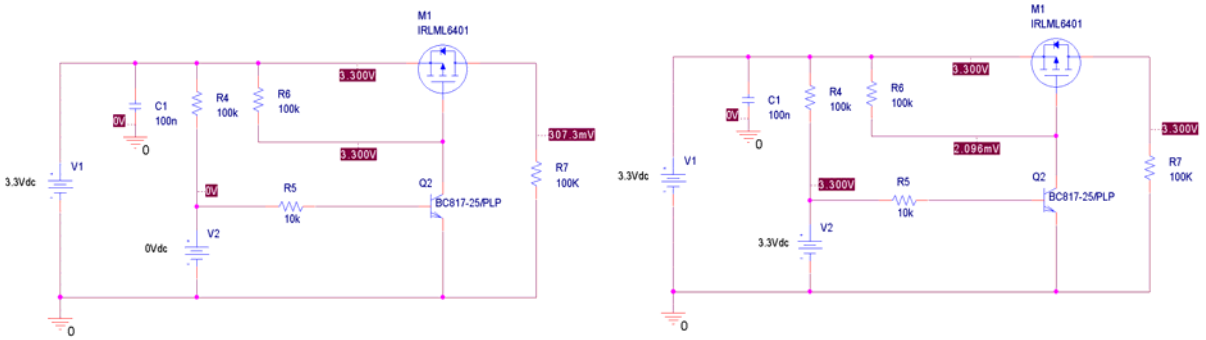


Figure 64 : Jumper JP5 opened.

8.6.2 Microcontroller

As said on section 6.6.1, the microcontroller is alimented either by VDD and VDDBU; moreover VDDA and GNDA which power the analogical part of the microcontroller are separate from VDD and GND. Actually those power sources are connected to the same alimentation plane but only to one point. The decoupling capacitors are presented on the right.

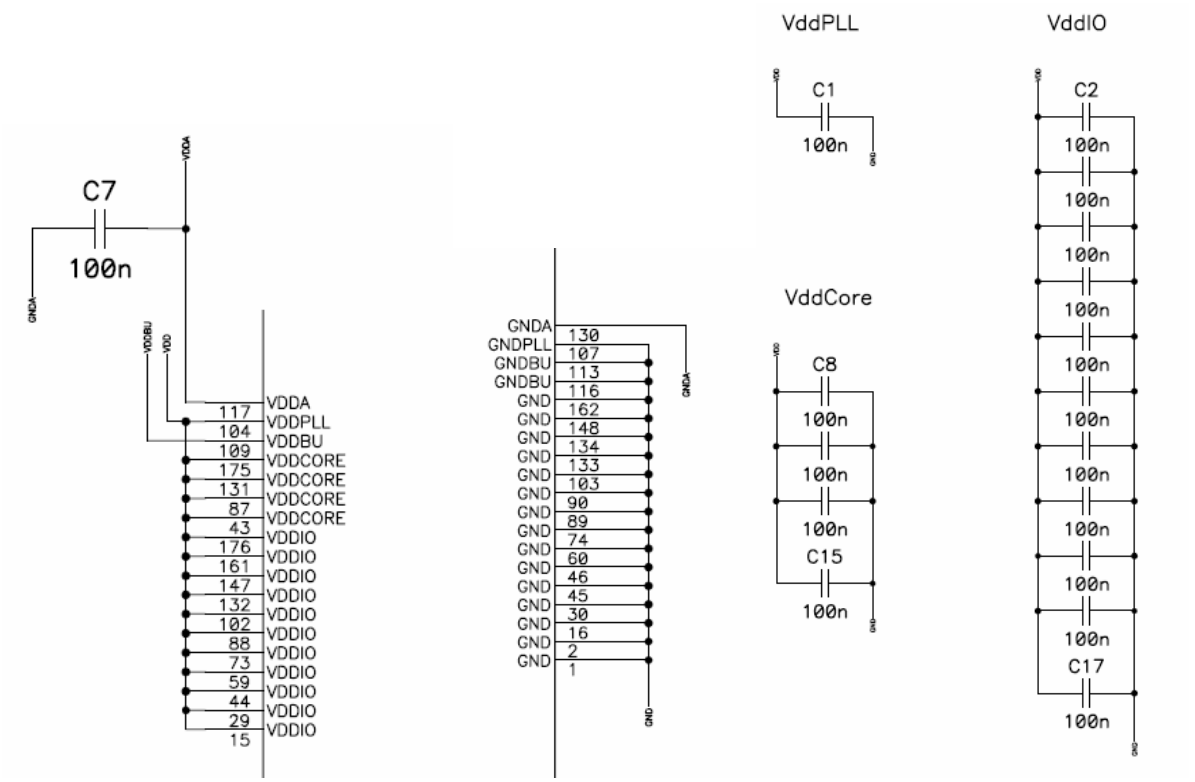


Figure 65 : Microcontroller alimentation

On the following figure different connections of the microcontroller are presented. For normal operation the pin NTRI must be help high during reset.

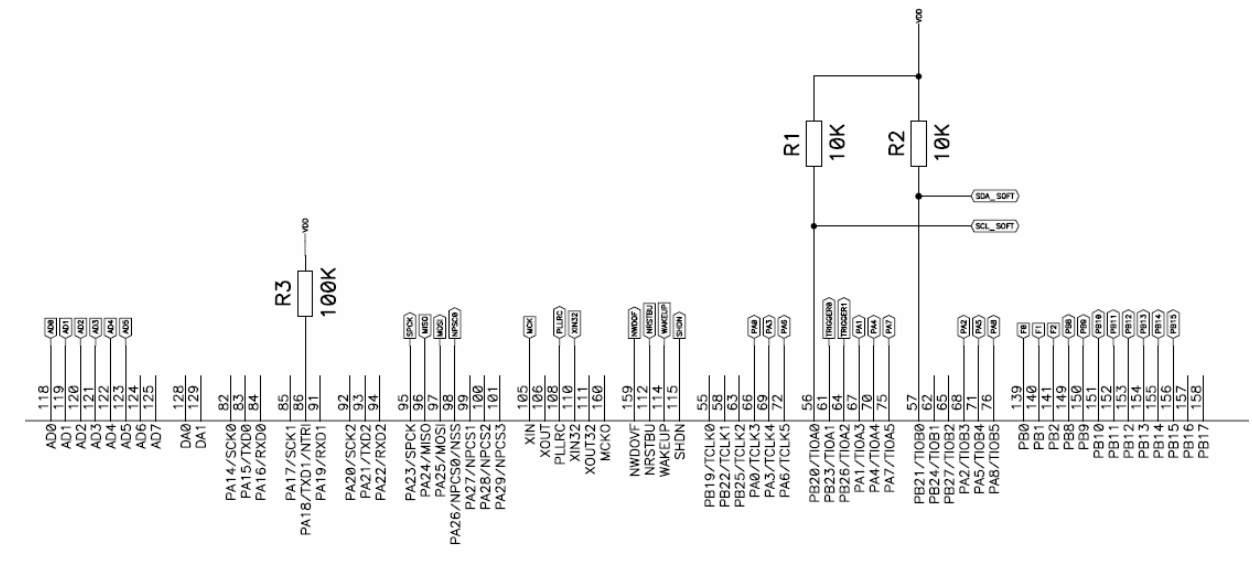


Figure 66 : Microcontroller connection

The NWAIT input can be used to add wait states at any time. NWAIT is active low and is detected on the rising edge of the clock. This functionality will not be used, so the pin is pulled up in order to deactivate it.

The BMS pin (Boot Mode Select) is sampled at reset. If it is detected high the external memory on NCS0 is 8-bit, if detected low 16-bit.

The JTAG debug mode is enabled when JTAGSEL is low.

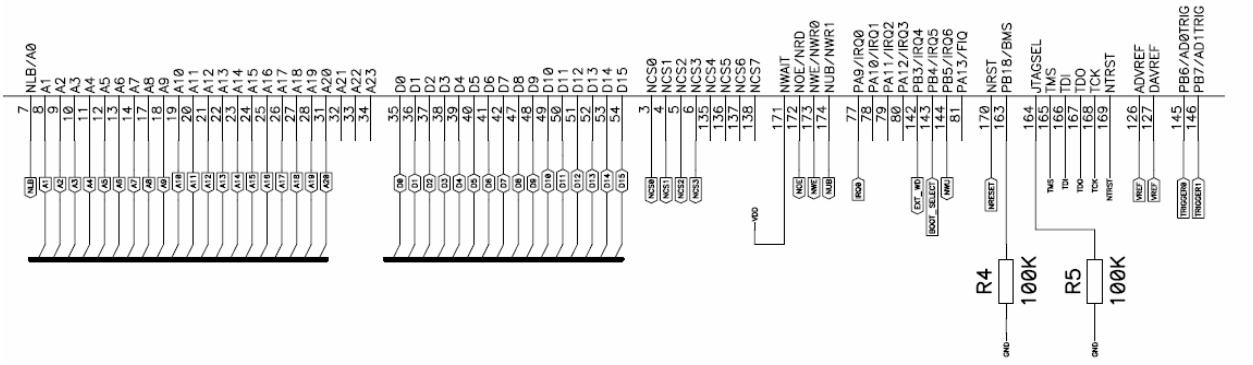


Figure 67 : Microcontroller connection

8.6.3 Memory

8.6.3.1 SRAM

The following table presents the pin description of the memory chip.

Pin name	Function
A0 to A17	Address input
I/O0 to I/O15	Data input/output
NCS1	Chip select 1
CS2	Chip select 2
NOE	Output enable
NEW	Write enable
NLB	Lower byte select
NUB	Upper byte select
Vcc	Power supply
Vss	Ground

The memory chip is connected as presented below. When CS2 is detected low the memory chip enters in an advanced sleep mode which draws 1[µA], for normal read / write access the CS2 must be high. When NCS1 is detected high, the value of CS2 does not care, the memory chip enters in a normal sleep mode which draws 4[µA]. In order to simplify the design CS2 is directly connected to Vcc and the advanced sleep feature is disabled.

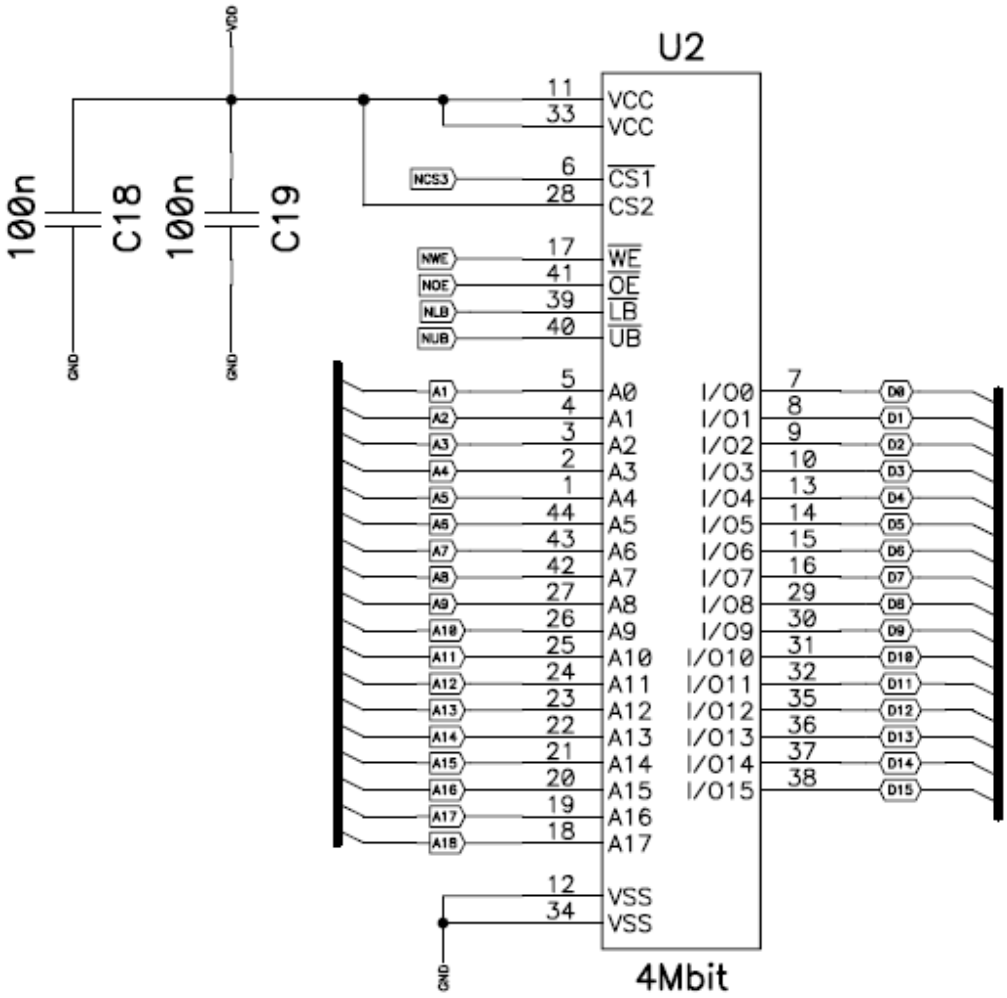


Figure 68 : SRAM connection

8.6.3.2 ROM

Vpp is used in order to program the memory chip. Once the chip is programmed Vpp may be connected directly to Vcc.

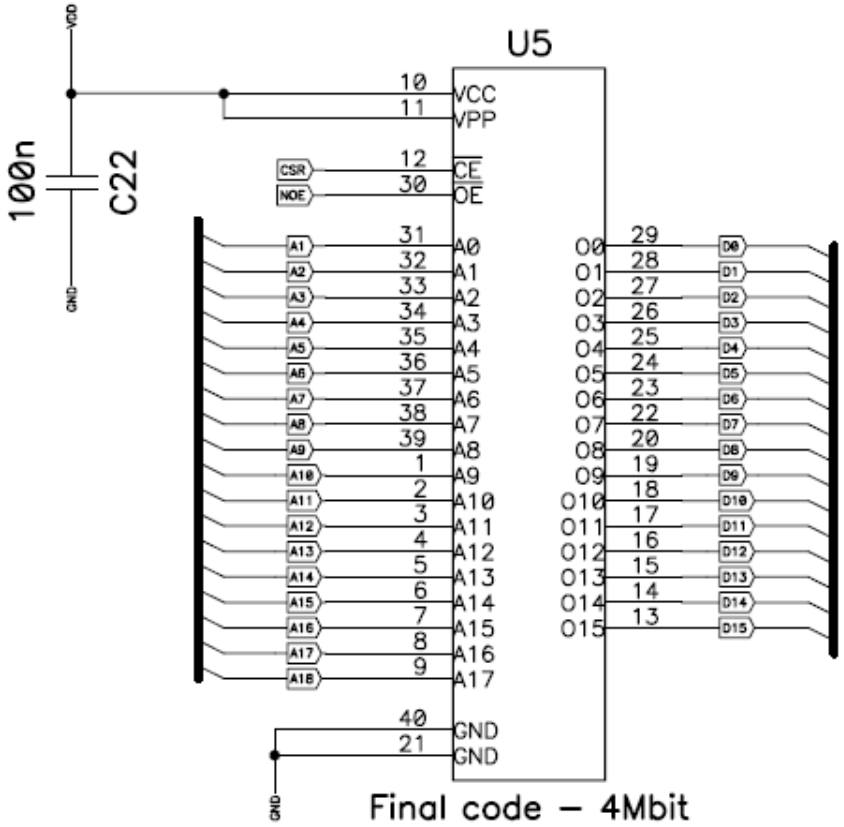


Figure 69 : ROM connection

8.6.3.3 Flash

Three different Flash memory chips will be installed on the board. One will be used to save the science data took by the payload, one in order to save command and other software uploaded from the earth and the last will be used only on the development board which will be unsoldered before the mission. The final flight software will be saved in the ROM. As the microcontroller is only able to boot from an external memory using the chip select 0, both the ROM and Flash must be connected to this chip select line. The jumper JP1 will be used to select on which memory the microcontroller will boot from. The two different chip select lines demultiplexed by the jumper are pull-up in order to guarantee the logic value if the two memory chip are soldered on same time.

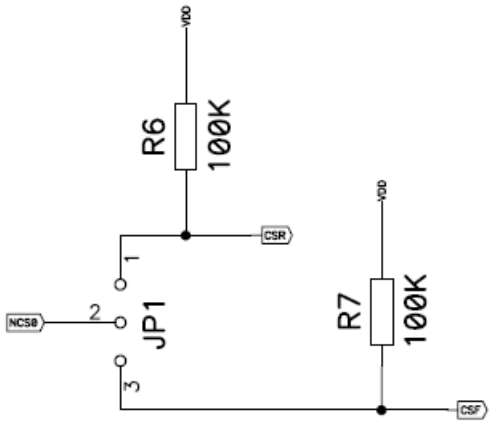


Figure 70 : Boot memory selection jumper

The following table presents the pin description of the flash memory chips.

Pin name	Function
A0 to A19	Address input
DQ0 to DQ14	Data input/output
DQ15/A-1	DQ15 (data input/output, word mode) A-1 (LSB address input, byte mode)
NBYTE	Selects 8-bit or 16-bit mode
NCE	Chip enable
NOE	Output enable
NRESET	Hardware reset pin
RY/NBY	Ready/Busy output
Vcc	Power supply
Vss	Ground

As the microcontroller offers the possibility to access external memory with a 16-bit data bus, the NBYTE pin is asserted by a high logic to configure the memory chip for 16-bit access.

The NRESET input pin is used to reset the internal state machine and not the memory content of the chip. To ensure that no spurious alteration of the memory content occurs during a malfunction, this pin will be asserted by the reset logic.

The RY/NBY pin output is connected to a GPIO of the microcontroller which offers the possibility to manage this function by software.

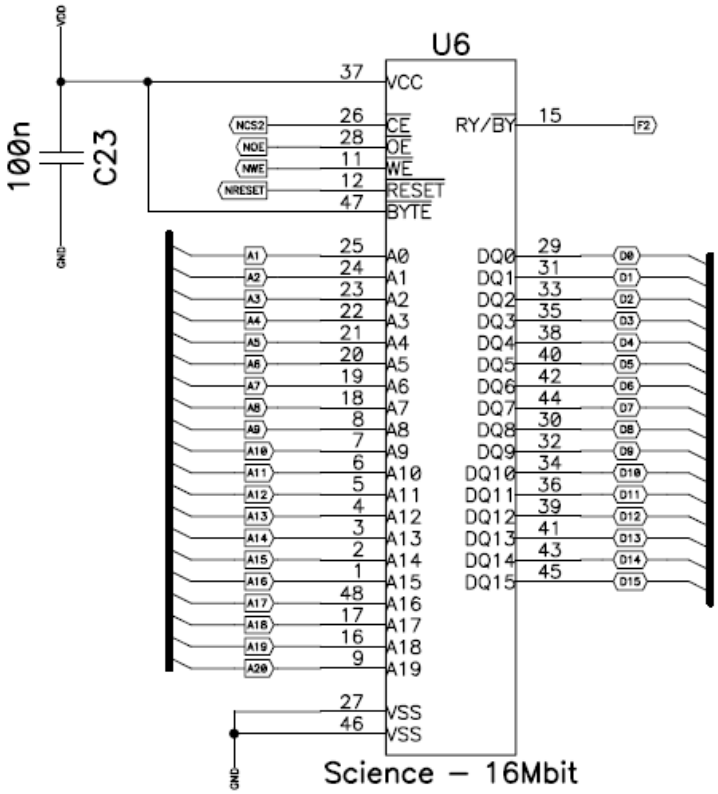


Figure 71 : Flash memory chip connection

8.6.4 Clock Sources

The microcontroller has three different clock sources. A 32,768[kHz] oscillator connected to the RTC. The microcontroller start on this clock in order to minimize the power required for the startup. A master clock input defined at 4[MHz] for this board, and finally a PLL which offers the possibility to multiply the master clock in order to reach another clock frequency. Atmel provide an Excel file [N5] to calculate the PLL filter.

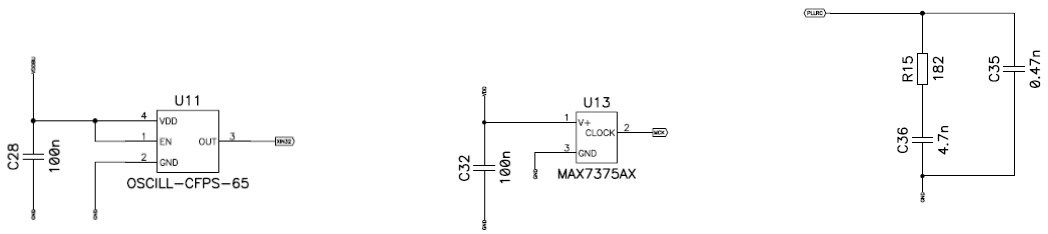


Figure 72 : Clock sources

Those three clock sources combined with the APMC provide the possibility to switch the frequency of the microcontroller.

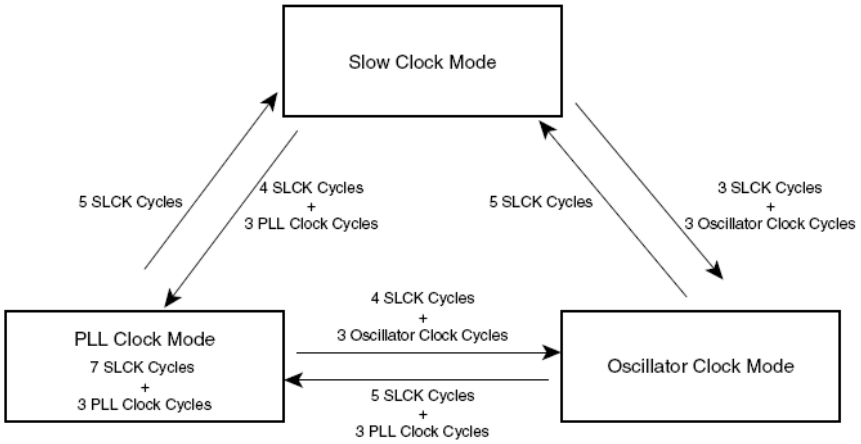


Figure 73 : Clock switching

8.6.5 Reset logic

During power transition, it is possible that the required minimum voltage supply is not respected, if this case happens it will be necessary to not power up the components so that a no given state appears. The MAX823 has this role, it compares the voltage supply with an internal reference and assert the reset pin if the voltage supply is not sufficient. The AT91M55800A has an internal watchdog which reset the microcontroller in case of overflow. This watchdog drives an external pin, NWDOF, which will be used in order to reset all the board.

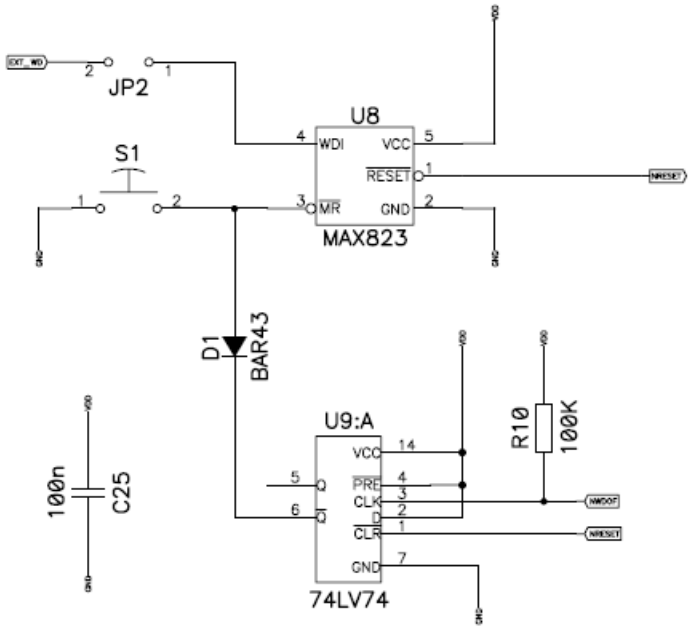


Figure 74 : Reset logic

8.6.6 Latch-up Protection

The development of the latch-up protection is referred to [R3].

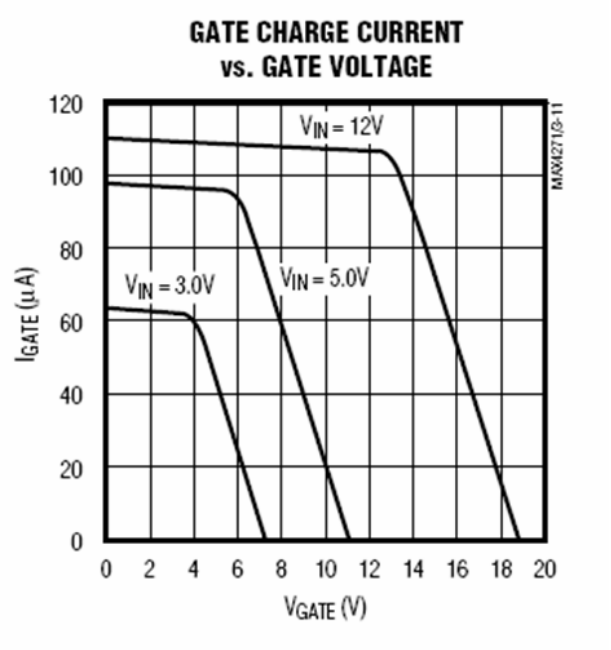


Figure 75 : Gate charge current vs. gate voltage

The following formula presents a dimensioning example. The maximum and minimum current drawn by the load must be known really precisely. As the CDMS board is not available at the time this section has been written, the current could not be measured and the dimensioning of the latch-up for the CDMS has been reported.

Given data: $I_{fast}; I_{slow}; C_L;\ t_L = \frac{C_L \times V_{in}}{I_{fast}}$

Programmable parameters:

$t_{NMOS,on} = \frac{C_{NMOS} \times 10V}{100\mu A}$;

$t_{on} \geq \max(t_L ; t_{NMOS,on})$;

$t_{on}(ms) = 0.31 \times CTON(nF)$;

$t_{off} = 32 \times t_{on}$; Attention, the time-constant is $\tau = 1k\Omega \times C_L$;

$t_{off} > \max(t_{on} ; \varepsilon \times 1k\Omega \times C_L)$; with $\varepsilon \geq 1$.

$t_{off}(ms) = 10 \times CTIM(nF)$;

The response time of the slow comparator is: $t_{slow} = \frac{CSPD \times 1.2V}{6\mu A}$; (20[µs] by default)

The response time of the rapid comparator is: $t_{fast} = 350ns$; (fixed value)

$$R_{sense} = \frac{50mV}{I_{slow}} \quad \text{and} \; R_{TH} = \frac{R_{sense} \times I_{fast}(mV)}{10} \; ;$$

The time for opening the NMOS in the case of an overload is: $t_{NMOS,off} = \frac{C_{NMOS} \times 10V}{200\mu A}$;

The time for opening the NMOS in the case of a short-circuit is: $t_{NMOS,off} = \frac{C_{NMOS} \times 10V}{1mA}$.

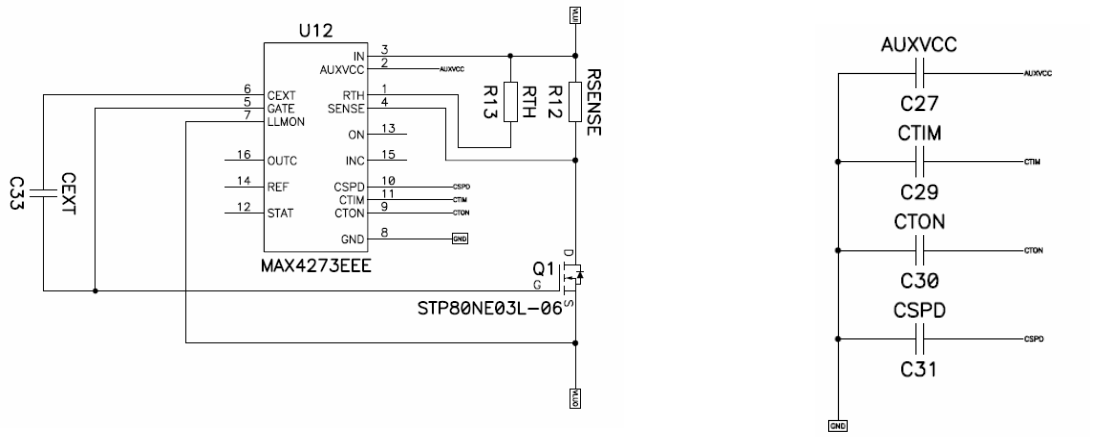


Figure 76 : Latch-up protection schematic

The resistor RSENSE (R12) allows to measure the current consumed by the user and to fix his threshold. R12 is a 2515 resistor which order of magnitude is 0,5[Ω] and allows a power dissipation of 1[W]. The other resistor and capacitor are 0603 in order to save place and weight.

8.6.7 SPI to I2C Bridge

The AT91M55800A does not provide a hardware I²C interface. It is possible to manage the I²C bus by software with GPIO, some libraries are provided by Atmel, and moreover the CDMS prototype board provides a bridge between the SPI internal controller and the I²C bus. The final selection between the hardware and software implementation of this bus will be decided by the Flight Software group. The following figure presents the implementation of the bridge.

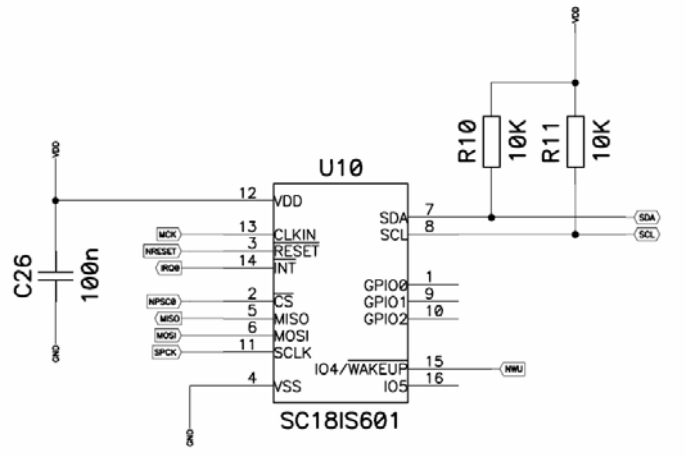


Figure 77 : SC18IS601

8.6.8 External Connector

The connectors available on the CDMS prototype board are presented in this section. At that time the final type of connector has not been decided.

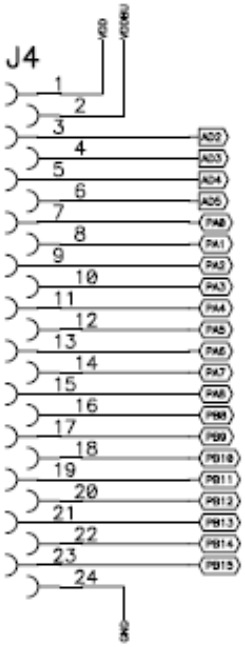


Figure 78 : Miscellaneous connector

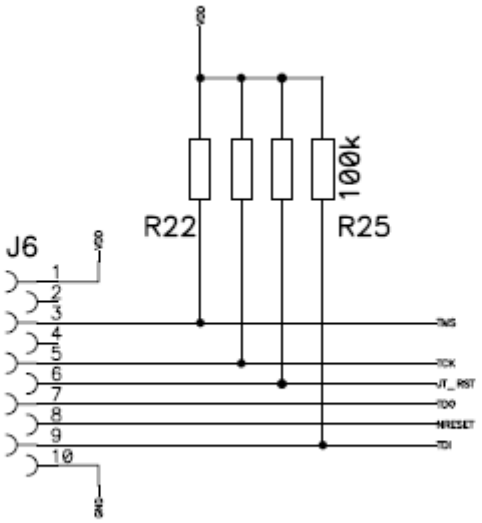


Figure 79 : JTAG connector

The jumpers JP6 and JP7 offer the possibilities to select either the software I^2C interface or the hardware I^2C interface. It has been said in section 6.2.3.2 that it should be interesting to test the possibilities to broadcast the master clock signal in order to save power, this signal is available with the pin 6 of this connector.

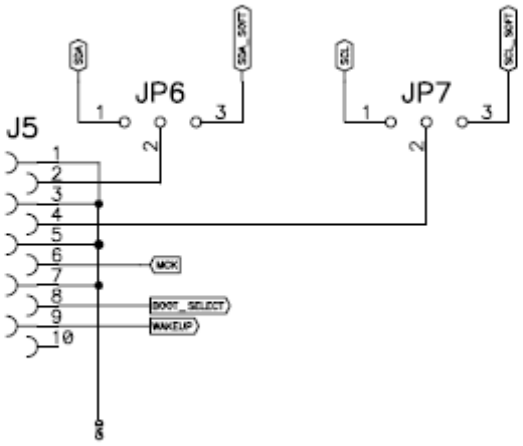


Figure 80 : Subsystem connector

8.6.9 Divers

Some others elements are not presented in this section. The temperature sensor, Vref voltage for analogical part of the microcontroller, RC circuit for the reset pulse of the APMC and some switch which emulate the function distributed by the others subsystems. The complete schematic is presented in Appendix C.

8.7 Bill of Material

Bill of material							
#	QTY	REFDES	DEVICE	VALUE	Distributo	Price [SFr]	QTY ordered
1	1	U1	AT91M55800A	CPU	Atmel	Sample	5
2	1	U2	R1LV0416	SRAM - 4Mbit	MSC/GE	Sample	2
3	3	U3, U4, U6	S29AL016	FLASH - 16Mbit	Farnell.com	6.20	6
4	1	U5	AT27BV4096	ROM - 4Mbit	Digikey.com	14.00	2
5	1	U7	74LVX08M	OR logic	Farnell.com	0.90	2
6	1	U8	MAX823	Supervisor	Farnell.com	4.75	2
7	1	U9	74LV74D	D flip-flop	Farnell.com	0.79	2
8	1	U10	SC18IS601	Bridge	Digikey.com	3.50	5
9	1	U11	CFPS-65	32.768[kHz]	Farnell.com	8.75	2
10	1	U12	MAX4273	Latch-up	Maxim	Sample	4
11	1	U13	MAX7375AX	4[MHz]	Maxim	Sample	5
12	2	U14, U15	DS60R	T° sensor	Farnell.com	4.10	4
13	1	U16	LM4040	Diode	Farnell.com	3.20	2
14	4	JP1, JP4, JP6, JP7	JUMPER	3 points	HEVs		
15	3	JP2, JP3, JP5	JUMPER	2 points	HEVs		
16	2	S1, S2	Switch		HEVs		
17	1	J1	Low voltage connector		HEVs		
18	1	J4	Header 24		HEVs		
19	2	J5, J6	Header 10		HEVs		
20	1	Q1	STP80NE03L-06	Mosfet	Farnell.com	9.95	2
21	1	Q2	IRLML6401	Mosfet	Farnell.com	0.90	5
22	1	Q3	BC817	NPN	Farnell.com	0.24	10
23	1	D1	BAR43	Schottky	Farnell.com	2.05	5
24	31	C1-C26, C28, C32, C34, C37, C39	Capacitor - 0603	100[nF]	HEVs		
25	2	C38, C40	Capacitor - 0805	10[uF]	Farnell.com	0.54	10
26	1	C35	Capacitor - 0603	0.47[nF]	HEVs		
27	1	C36	Capacitor - 0603	4.7[nF]	HEVs		
28	5	C27, C29, C30, C31, C33	Capacitor - 0603	TBD			
29	5	R1, R2, R11, R12, R19	Resistor - 0603	10[kΩ]	HEVs		
30	14	R3 - R10, R17, R18, R21, R22 - R25	Resistor - 0603	100[kΩ]	HEVs		
31	1	R20	Resistor - 0603	200[kΩ]	HEVs		
32	1	R13	Resistor - 2515	TBD			
33	1	R14	Resistor - 0603	TBD			
34	1	R15	Resistor - 0603	182[Ω]	HEVs		
35	1	R16	Resistor - 0603	4.7[kΩ]	HEVs		
36	2	R26, R27	Not populated				

Some manufacturer or distributors have send samples of components. The price for one CDMS prototype board, including the price of the samples, is expected to be lower than 150[SFr] only for the components.

8.8 PCB layout

The layout of the PCB has been made with P-CAD 2006 from Altium.

This layout has been made conforming to the reference [R4]; the ECSS-Q-70-11A standard. This standard defines the requirements imposed on both the customer and the qualified PCB supplier for PCB procurement. This standard also details the basic requirements for procurement of PCBs as well as the minimum requirement for the different type of PCBs.

This standard is applicable for the following type of boards:

- Rigid printed boards (single-sided, double-sided, multilayer, sequential-laminated multilayer and metal core)
- Flexible printed boards (single-sided and double-sided)
- Rigid-flex printed boards (multilayer and sequential-laminated multilayer)
- High frequency printed boards
- Special printed boards

The following pages present the specification for a rigid multilayer board.

9.5 Rigid multilayer printed boards

These are the minimum requirements acceptable for qualification and procurement of rigid multilayer PCBs.

Table 4: Limits of approval and characteristics of finished rigid multilayer printed boards

Item	Limits
Base materials (according to ECSS-Q-70, IEC specifications and IPC 4101)	- Woven-glass-reinforced epoxy resin - Woven-glass-reinforced polyimide resin - Woven-glass-reinforced bismaleimide/trazine modified epoxy (HTg) resin - Non-woven aramide-reinforced polyimide resin
Dimensional characteristics	
External dimension tolerance	±0,2 mm
Thickness tolerance	±10 %
Maximum active board size	TBD by the supplier
Maximum board thickness	3,2 mm
Positioning between registration mark and edge of circuit	±0,2 mm
Conductor width	
- internal	120 µm minimum
- external	200 µm minimum (for fine pitch 120 µm width is tolerated if less than 5 mm from component pad)
Conductor spacing	
- internal	150 µm minimum
- external	300 µm minimum (for fine pitch 150 µm spacing is tolerated if less than 5 mm from component pad)
Conductor tolerance (minimum/maximum)	TBD by the supplier
Tolerance on diameter of terminal pads	TBD by the supplier
Minimum drilled hole diameter	
- component hole	According to ECSS Q-70-08
- via hole	0,25 mm minimum and maximum aspect ratio t/d = 6
Tolerance on diameter of plated-through holes	
- nominal Ø ≥ 0,7	Δ maximum 0,15 mm for component hole
- nominal Ø < 0,7	Δ maximum 0,20 mm
Tolerance on diameter of non-plated-through holes	Δ maximum 0,20 mm
Positioning of holes with respect to reference mark	±0,1 mm
Relative misregistration pad/hole	≤ 0,15 mm
Misalignment determined by measuring minimum annular ring	
- external layers	
• solder side	0,20 mm
• component side (reduced pads)	0,10 mm
• non-soldering hole	0,10 mm
- internal layers	50 µm

37

Table 4: Limits of approval and characteristics of finished rigid multilayer printed boards *(continued)*

Item	Limits
Layer to layer registration	±100 µm
Number of layers	18 maximum
Electrolytic coatings	
Electrolytic copper plating	
- minimum purity	99,5 %
- thickness	
• surface pattern	≥ 25 µm
• plated-through holes	≥ 25 µm
• via holes	≥ 20 µm
Tin lead plating after reflow	
- tin content of alloy	(63 ± 8) %
- thickness on surface	≥ 8 µm in highest part
- thickness in plated-through holes	≥ 8 µm in highest part (minimum half height of hole wall)
- on corner angle	≥ 2 µm
Electrolytic gold plating	
- minimum purity	99,8 % (shall not contain more than 0,2 % silver)
- thickness on nickel	(4 ± 3) µm
- thickness on copper	(5 ± 2) µm
Electrolytic nickel plating	Optional under gold
- thickness	2 µm to 10 µm
Insulation between layers	70 µm minimum
Mechanical characteristics	
Warp and twist	≤1,1 % for board thickness ≥ 1,6 mm ≤1,5 % for board thickness < 1,6 mm
Conductor adhesion/peel strength	
- on epoxy with Tg < 160 °C	≥ 16 N/cm
- on epoxy with Tg > 180 °C	≥ 12 N/cm
- on polyimide	≥ 12 N/cm
- on bismaleimide/trazine modified epoxy HTg	≥ 12 N/cm
- aramide/polyimide	≥ 6 N/cm

Table 4: Limits of approval and characteristics of finished rigid multilayer printed boards *(continued)*

Item	Limits
Bond strength/pull strength	
- for terminal pads 4 mm Ø	
• on epoxy Tg < 160 °C	≥ 140 N
• on epoxy Tg > 180 °C	≥ 80 N
• on polyimide	≥ 80 N
• on bismaleimide/trazine modified epoxy HTg	≥ 60 N
• on aramide/polyimide	≥ 60 N
- for terminal pads 2 mm Ø	
• on epoxy Tg < 160 °C	≥ 35 N
• on epoxy Tg > 180 °C	≥ 20 N
• on polyimide	≥ 20 N
• on bismaleimide/trazine modified epoxy HTg	≥ 12 N
• on aramide/polyimide	≥ 12 N
Electrical characteristics	
Insulation resistance	
- intralayer	> 10^4 MΩ
- interlayer	> 10^5 MΩ
Withstanding voltage per mm spacing between conductors	
- intralayer and interlayer	1 000 V r.m.s.
Short time overload	
- 35 µm copper thickness	7 A for 4 s
- 70 µm m copper thickness	14 A for 4 s
Long time overload, destructive current	
- 35 µm copper thickness	I ≥ 8 A
- 70 µm copper thickness	I ≥ 16 A
Internal short circuit	
- insulation resistance	≥ 10^3 MΩ

The final layout of the CDMS prototype board is a six-layer, double-sided rigid printed board. The following figure is an overview of the layout of the CDMS board. All the components that will be present on the final board are implemented in the area of 40*90[mm]. Some other components like jumper or connector which will only be on the prototype has been placed around the area of 40*90[mm]. The layers are presented in Appendix D.

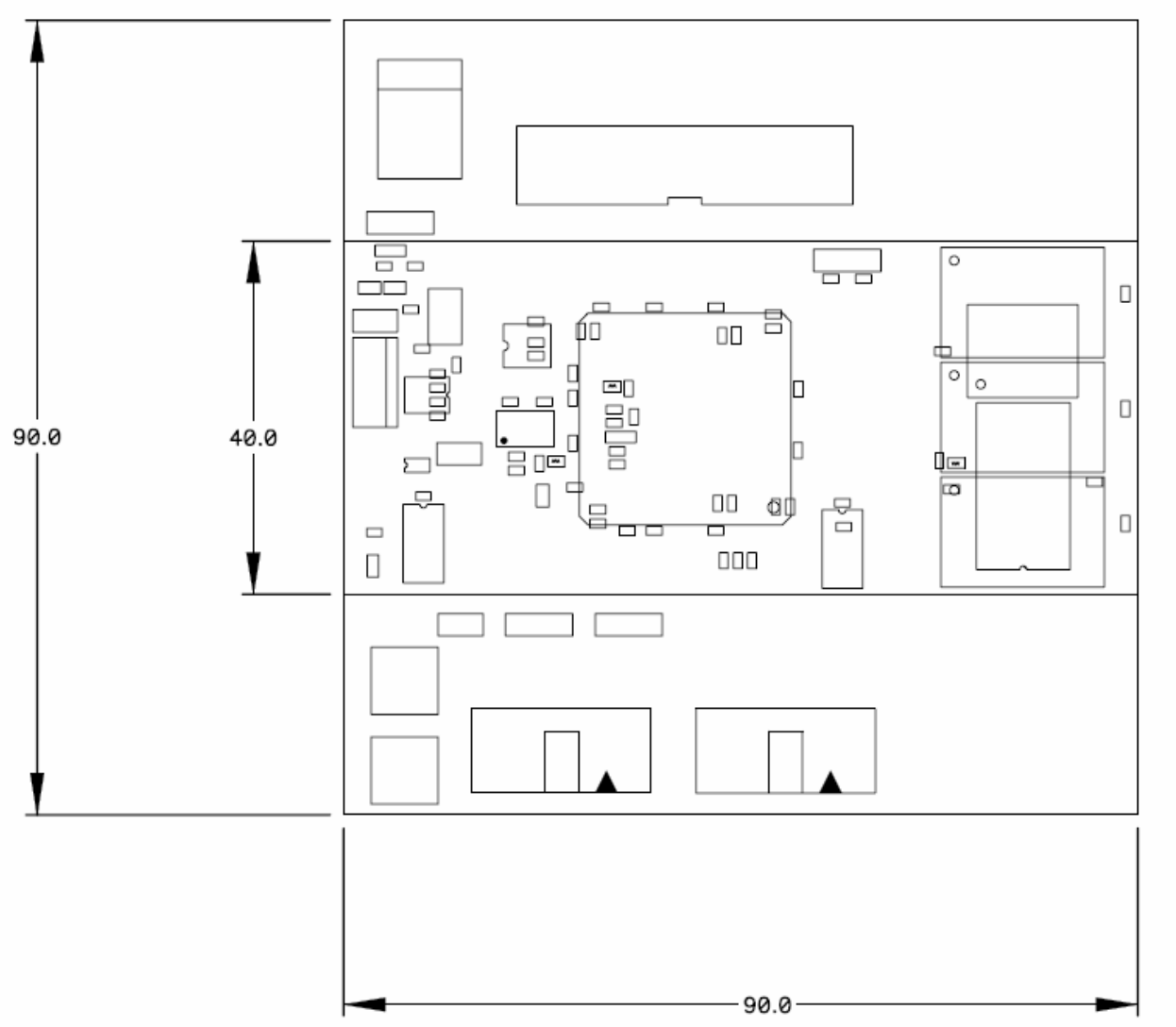


Figure 81 : PCB layout

The layer stack-up is presented below:

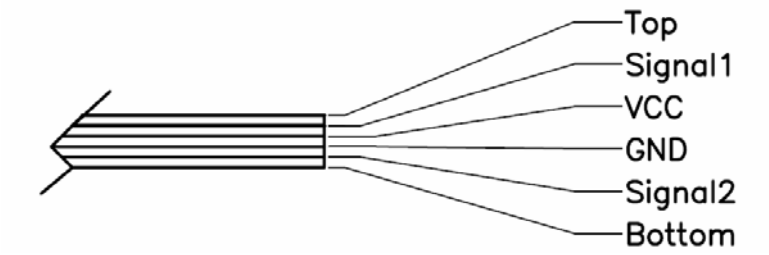


Figure 82 : Layer stack-up

The following figure presents the implementation of the interface with their explanation.

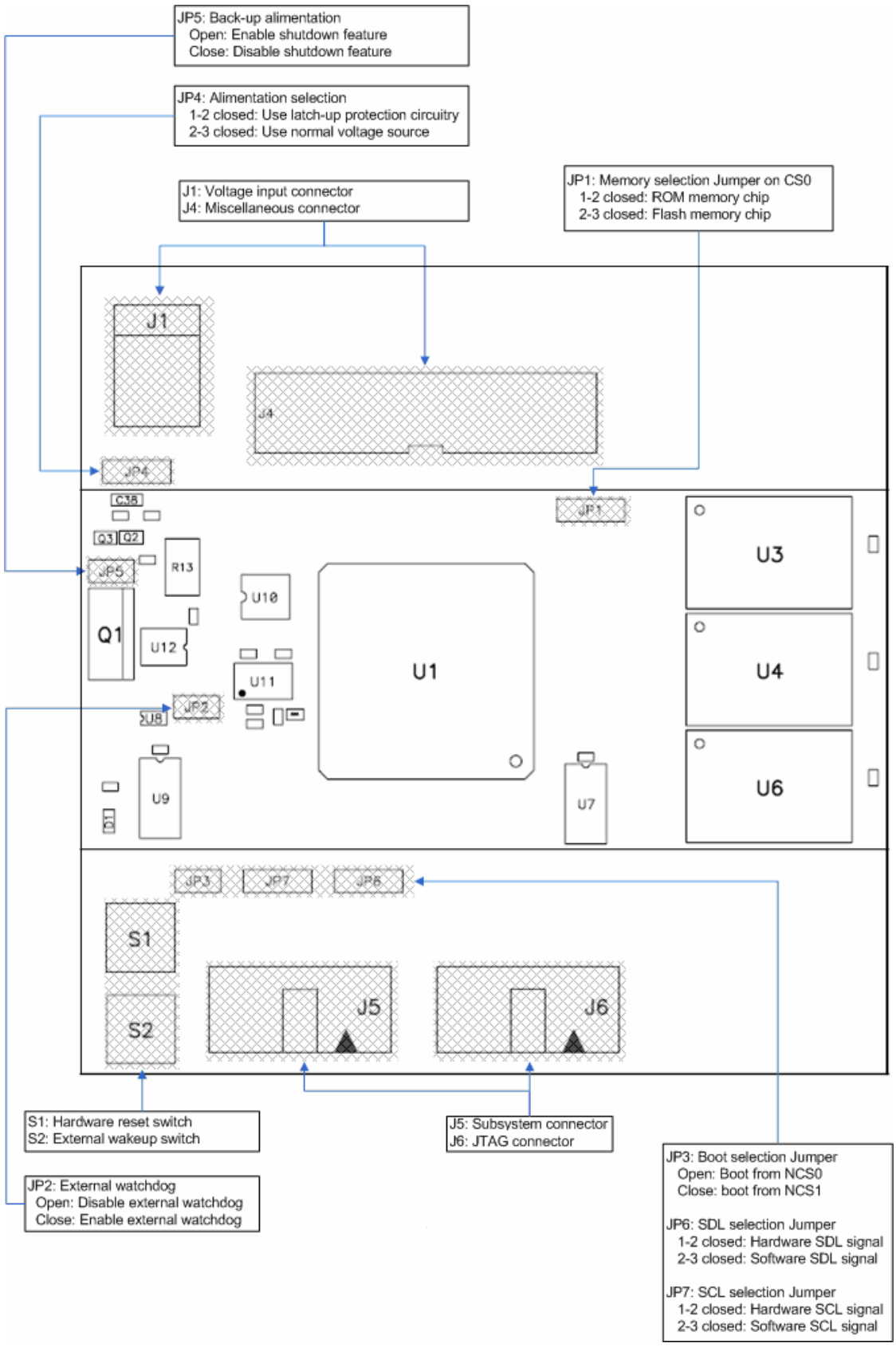


Figure 83 : Interface implementation

8.9 Analysis

The board developed at that time is only a prototype. This prototype will be used in order to validate the functional capacities of the CDMS, the choices which were taken, and to have a precise budget of the board. In order to optimize the board some analysis must be done. Those analysis and their goals are presented below.

8.9.1 Radiation Analysis

It means of analyzing the effect of the irradiative environment on the electronic components.

- To quantify the effects on the board.
  - On the analogical performances
  - On the numerical performances

The goal is to envisage possible software or hardware protections. For example: a latch-up protection.

8.9.2 Reliability Analysis

This analysis determine the lifetime of the board, it can be done using FIDES standards (based on the evaluation of 10 board from 100 to 5000 components). For the space field the analysis of reliability is based on document [R5] MIL-HDBK-217F. Two methods are used: part count and part stress.

The goal is to allow making modifications of the components not having the lifetime required by the project.

8.9.3 Thermal Analysis

This analysis determines the operation of the components within the limits of temperature given.

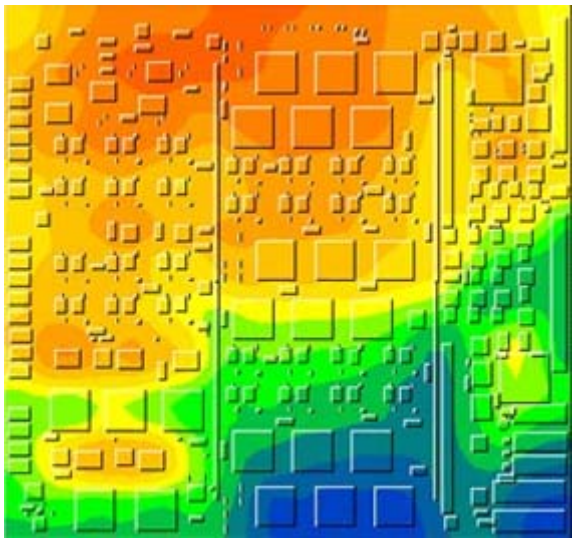


Figure 84 : Thermal analysis example

This analysis can help to solve problems of temperature of the components by indicating hot or cold places. It will also make it possible to see whether the limits of operations of the components can be exceeded in which case it would be necessary to change them.

8.9.4 Mechanical Analysis

This analysis shows the effects of the various mechanical constraints (acceleration, vibration) on the board and the components.

This analysis will provide the ability to check the solidity of the cards, fixations and soldering, to emphasize possible problems of placement of the components.

8.9.5 FMECA (Failure Modes, Effect and Critical Analysis)

This analysis makes the list of failure modes for each sub functions determined in the functional analysis, then, to present the data according to a standardized format, by indicating the effects of the malfunction on the higher systems, the probable gravity and causes.

Goals are to optimize the board, remove not reliable elements, and envisage a possible automatic covering of the breakdown.

8.9.6 PSA (Part Stress Analysis)

This analysis checks that the functional parameters of the components are respected with a safety margin to avoid stress and detect the components which reach or exceed the limits of the manufacturer.

With this analysis it is possible to estimate the degradation of the components as well as the fall of their limits and change the components exceeding the limits.

8.9.7 WCA (Worst Case Analysis)

This analysis makes it possible to evaluate the parameters of performance of the board considering the limiting values (worst case) of the components.

To identify starting from the circuit the values worst cases:

- Analogical
- Digital
- Timing

Compare these values with those given in the specification in order to check the ranges of the tolerances with a margin, to avoid errors and dysfunctions by replacement of unsatisfactory components.

9 SOFTWARE DEVELOPMENT

9.1 Development environment installation and configuration

This section will speak about the installation and configuration of the development environment that will be used for the CDMS project.

On the following figure are presented the various elements necessary to place software in an embedded system. The following elements can be distinguished: The PC with a cross development environment, an embedded system, in this case the target is the CDMS, and an interface use to establish the communication between the PC and the embedded system.

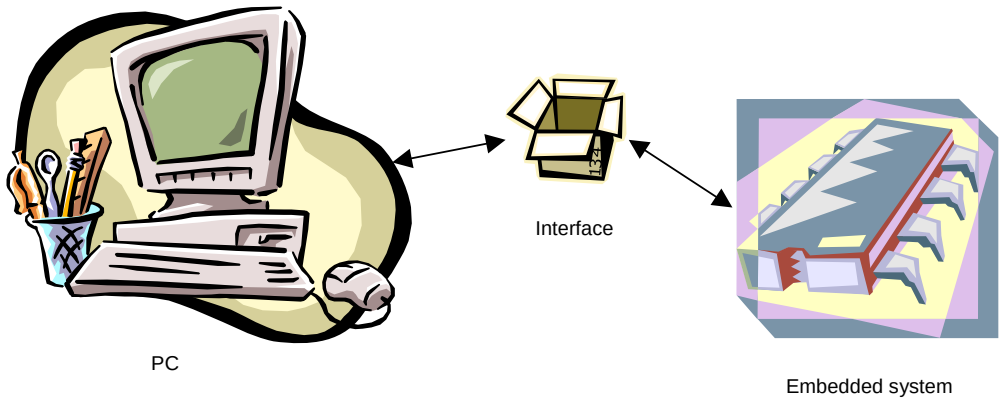


Figure 85 : Development architecture

A software development kit (SDK) is typically a set of development tools that allows a software engineer to create applications for a certain software package, software framework, hardware platform or similar. For this project it has been decided to use free software application in order to program the CDMS target. In that way Amontec created a complete solution for C and C++ development for the ARM processors, including installers for a graphical IDE, a compiler, a debugger, binary utilities, libraries, an on-chip debug and a JTAG solution.

The Amontec sdk4arm comes pre-compiled and native for Microsoft win32 systems (W2k, XP), and comes with installers. The version that will be used is the "20060730".

The Amontec sdk4arm includes four independent modules called:

- **Ide** module: integrated development environment

  Including Eclipse platform, Zylin CDT …

- **Gat** module: gnu arm toolchain

  Including GNU GCC, GDB, Insight, Make, Utils …

- **Ocd** module: on-chip debug

  Including OpenOCD, JTAGkey Drivers & Demo …

The Amontec sdk4arm should be downloaded from: http://www.amontec.com/sdk4arm.shtml.

Once the compressed file is downloaded, unzip the Amontec-sdk4arm-yyyymmdd-win32.zip file (in this case yyyymmdd is 20060730) in a temporary folder. Run the included Amontec-sdk4arm-install.exe application and install all the modules.

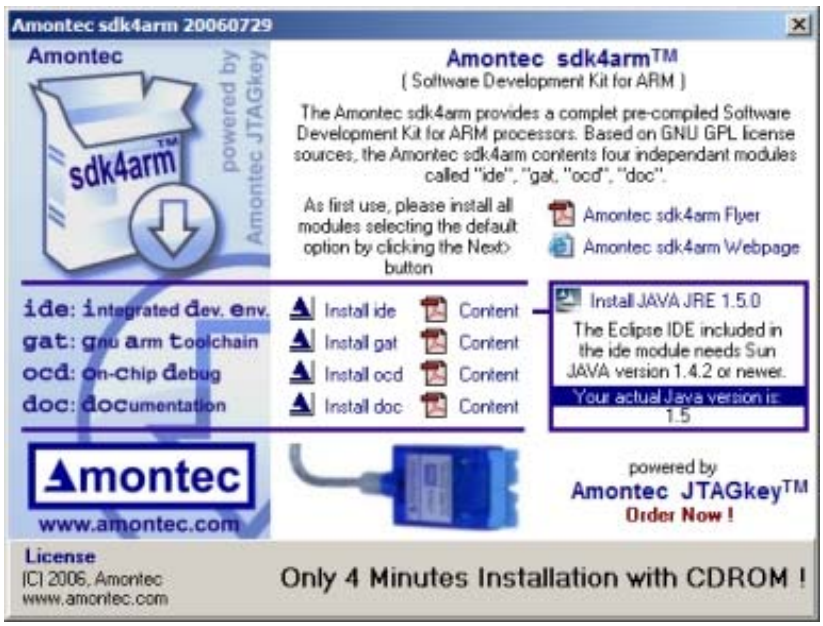


Figure 86 : Amontec installation interface

The amontec-sdk4arm-install.exe application helps to install all modules, and helps to install or update your SUN JAVA virtual machine used by Eclipse Platform. The installation will be performed with typical configuration. Before continuing check that the following lines are present in Control Panel > System > Advanced > Environment variables > System variables > Path:

%SystemRoot%\system32;

C:\Program Files\amontec\sdk4arm\gat\garm\bin;

C:\Program Files\amontec\sdk4arm\gat\utils\bin;

C:\Program Files\amontec\sdk4arm\ocd\openocd\bin

The OpenOCD precompiled version of the SDK4ARM will not provide the support for programming through the parallel port but only with the JTAGkey. Depending on the debug interface available it will be necessary to install the version which supports the parallel port interface.

For this project the chameleon pod from Amontec will be used and will be programmed in wiggler.

The Open On-Chip Debugger has been created by Dominic Rath as a part of a diploma thesis at the University of Applied Sciences, FH-Augsburg. The installer for OpenOCD can be downloaded here: http://www.yagarto.de/download/openocd/openocd-2006re100-setup-rc01.exe.

The version used in this project is the 2006re100. After the installer has been started, it is possible to select the features that must be installed at the "Choose Components" page:

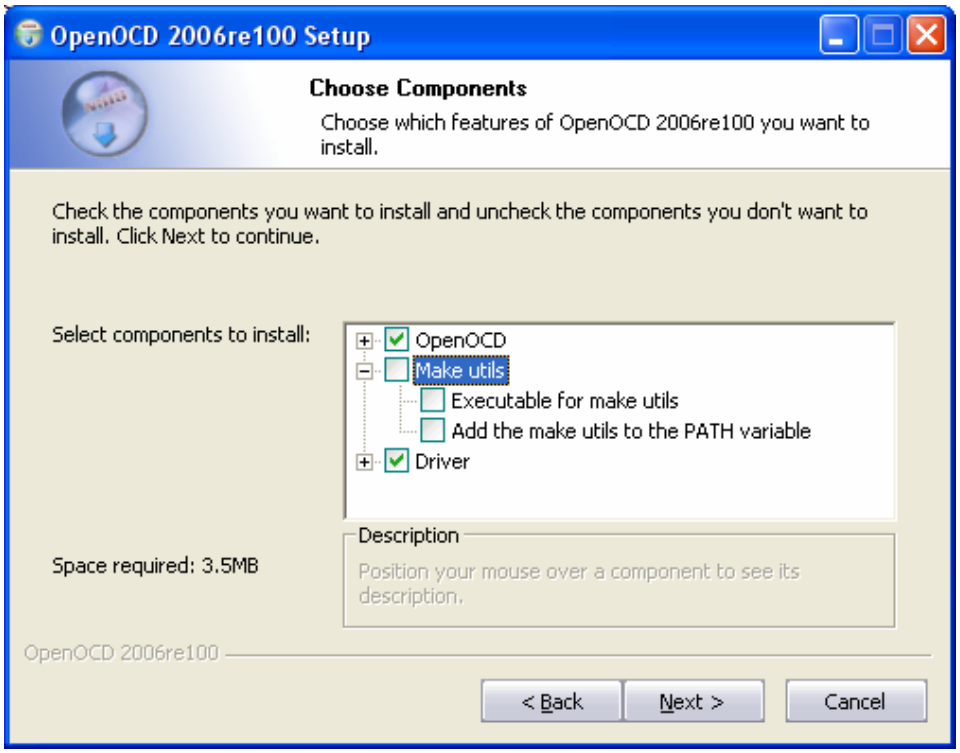


Figure 87 : OpenOCD "Choose Components" window

As the "Make utils" are already installed with the SDK4ARM from Amontec, unselect the corresponding checkmark. OpenOCD will be installed in the same folder as SDK4ARM.

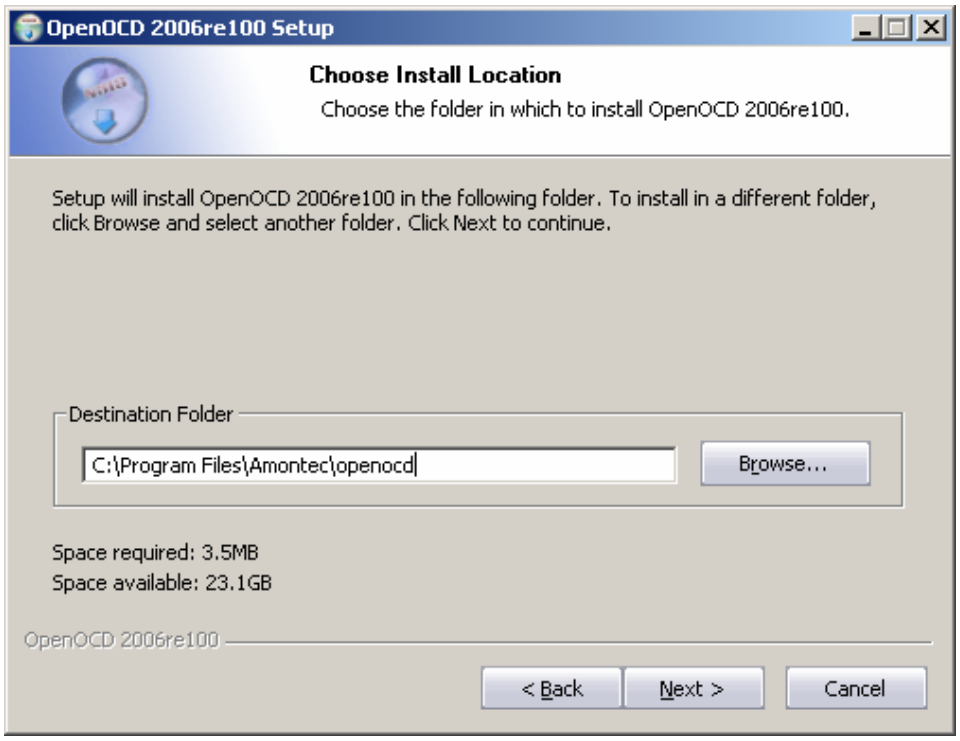


Figure 88 : OpenOCD installation location

After the end of the installation the driver must be installed. If the checkmark at the driver section above was left, the driver for the JTAGKey and for the parallel port has been installed on your PC. Now the driver must be loaded, it is possible the load either the JTAGKey and/or the parallel port driver. For the parallel port driver open a command prompt and go to the following directory:

"C:\Program Files\Amontec\openocd\driver\parport"

The driver will be loaded with "install_giveio" and can be unloaded with "remove_giveio".

The following figure presents the procedure to load the driver and how the output should look like.

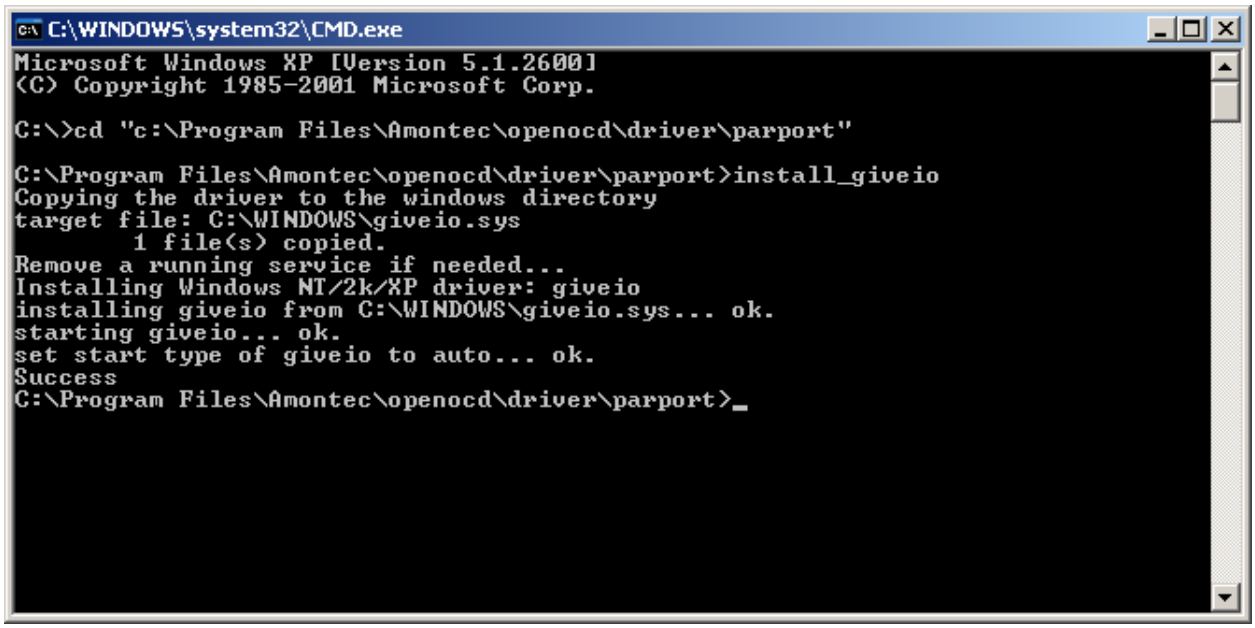


Figure 89 : Parallel port driver installation

When installing the sdk4arm, the installer adds the path of the most important tools in the PATH environment variable. But the installer adds a very powerful new Environment variable: %SDK4ARM% which represents the folder where sdk4arm has been installed. When using this variable in Eclipse, make sure to use %SDK4ARM% and not %sdk4arm%. Eclipse is based on SUN JAVA which one is a bit case sensitive.

It is now possible to check the version of each most important package as the GNU GCC Insight Toolchain and the OpenOCD JTAG server provided by the SDK4ARM from. Open a Windows Command Prompt and type the following command.

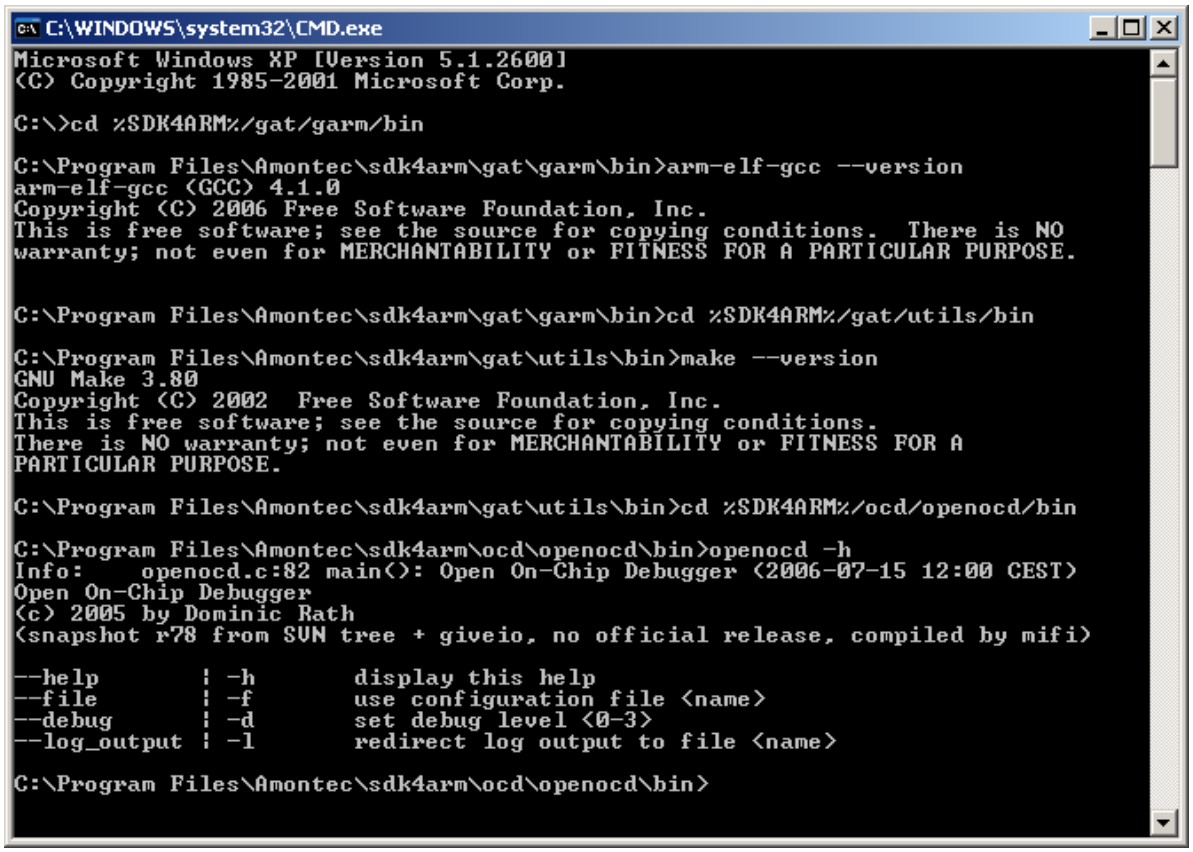


Figure 90 : GNU tools installation verification

The availability of the OpenOCD and driver for parallel port can be checked by making a short test. Those tests should be made by typing the following command in a Windows Command Prompt.

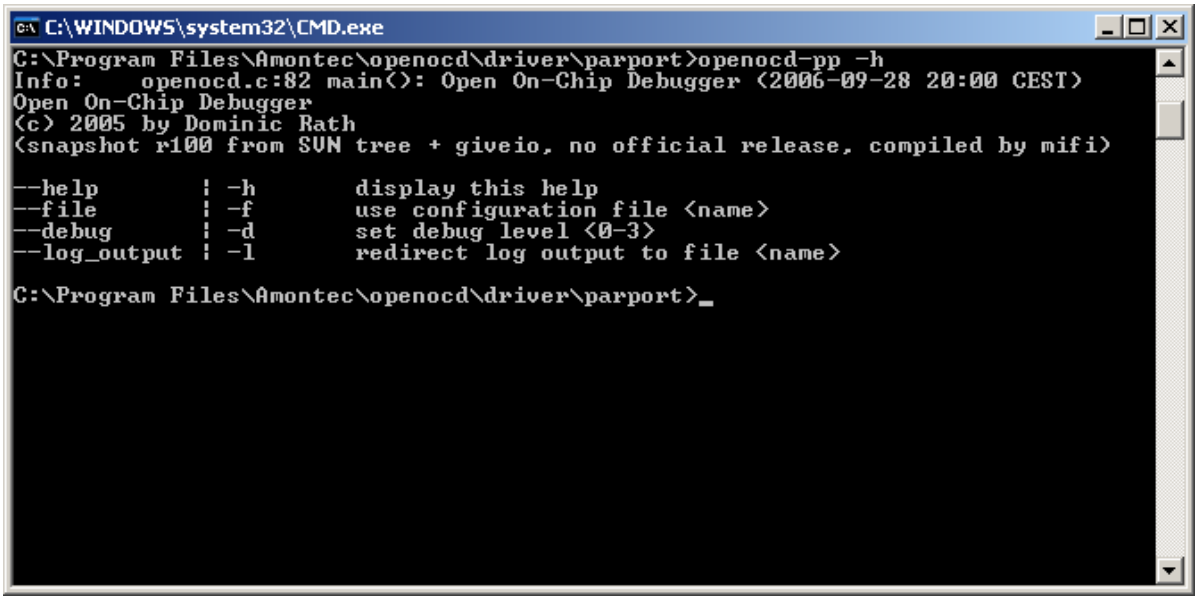


Figure 91 : OpenOCD installation verification

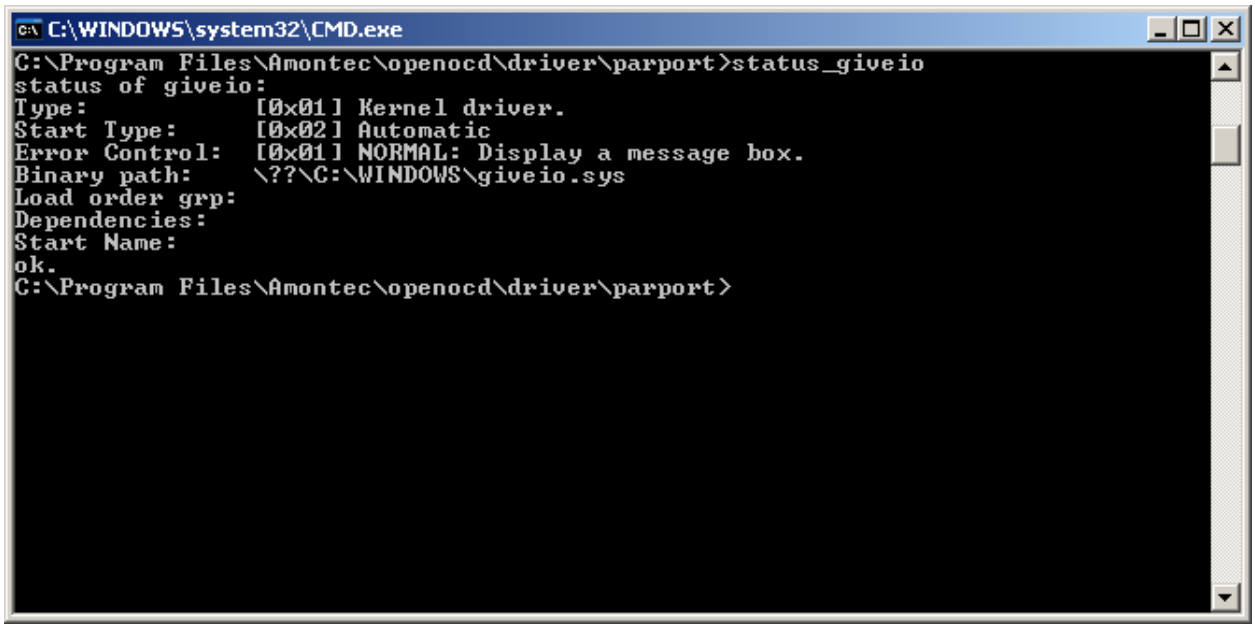


Figure 92 : Parallel port driver installation verification

At that time the installation of the development environment is completed. As said before the interface used for this project is provided by Amontec and is named "Chameleon". This chameleon will be programmed in wiggler.

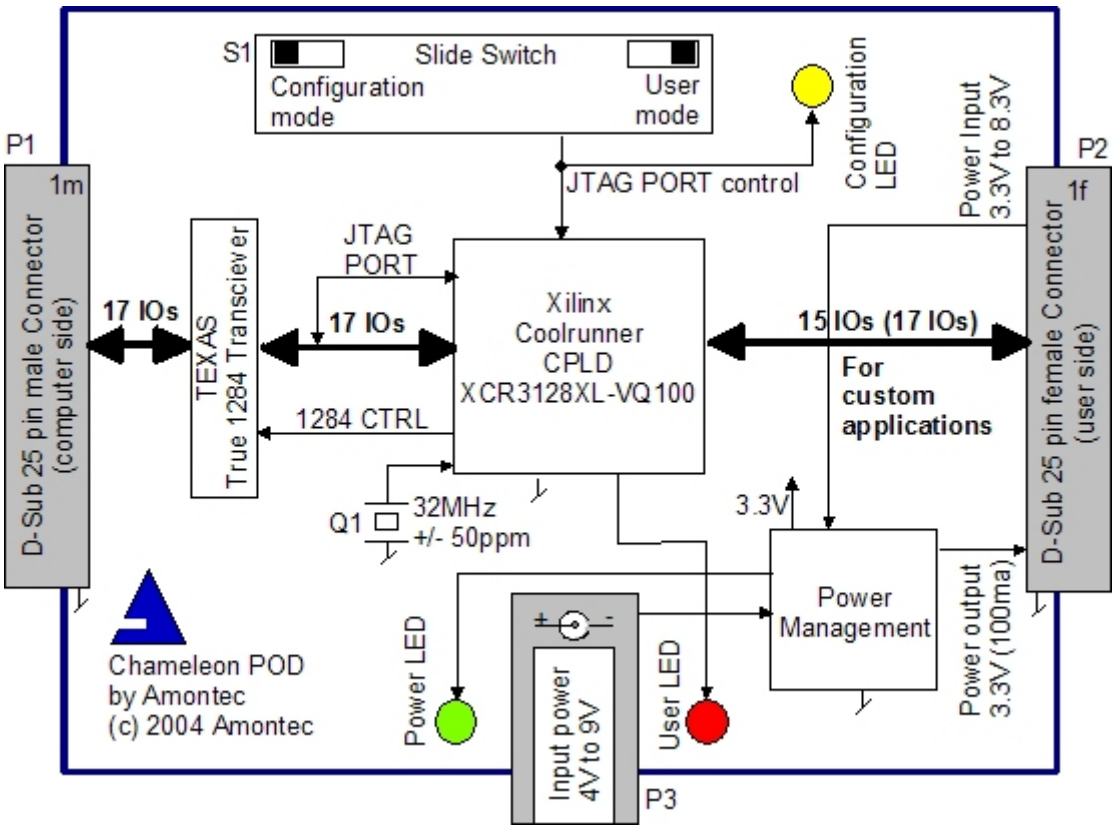


Figure 93 : Chameleon block diagram

The configuration will be presented now. Some files are required to do this operation, the "Chameleon programmer", specific drivers for parallel port, and the configuration files for the CPLD and should be downloaded from those following addresses:

Chameleon programmer:

http://www.amontec.com/fix/project/chameleon/appl/amt_chm_programmer_setup_10.exe

Parallel port driver:

http://www.amontec.com/fix/project/chameleon/appl/amt_parallelport_driver_setup.exe

Configuration file:

http://www.amontec.com/fix/project/chameleon/config/chm_appl_jtag_wiggler.amtsvf

Install the parallel port driver by double clicking on the executable file then install the Amontec Chameleon programmer with the default options.

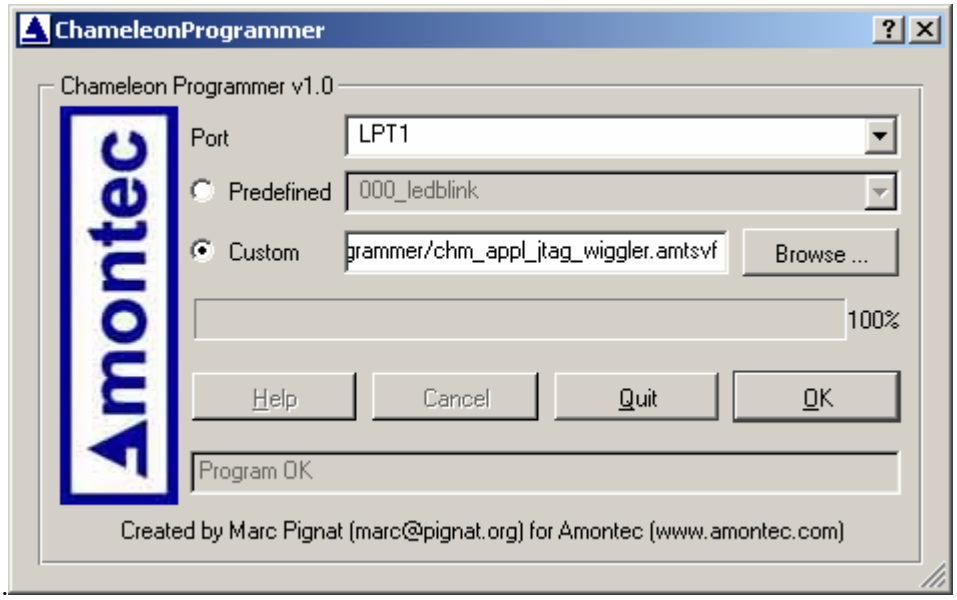


Figure 94

Put the hardware switch of the Chameleon in "Configuration mode", then launch the Chameleon programmer and select the configuration file downloaded further. Then press "OK" in order to program the Chameleon. Once the programming is performed, put the switch in "User mode". It is required to have the chameleon connected to the parallel port and powered for this operation.

At that time the installation is done and it is time to start with eclipse. The Appendix E will shortly present how to use Eclipse.

9.2 Open source project architecture

This section presents the necessary files required for the development of the CDMS with eclipse. The Source code is presented in Appendix F. The following figure shows how a compiler driver works.

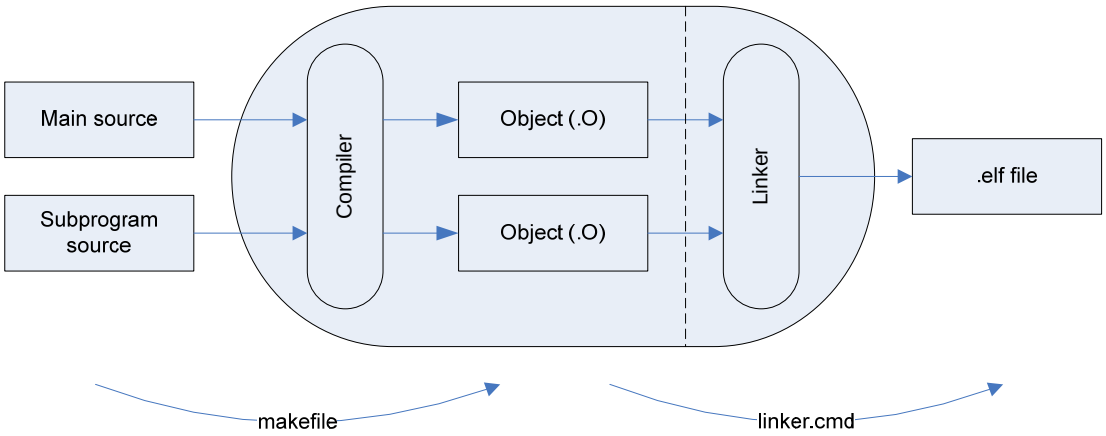


Figure 95 : Compiler driver

In the case of the CDMS project the files required in order to make test software are presented in the following figure.

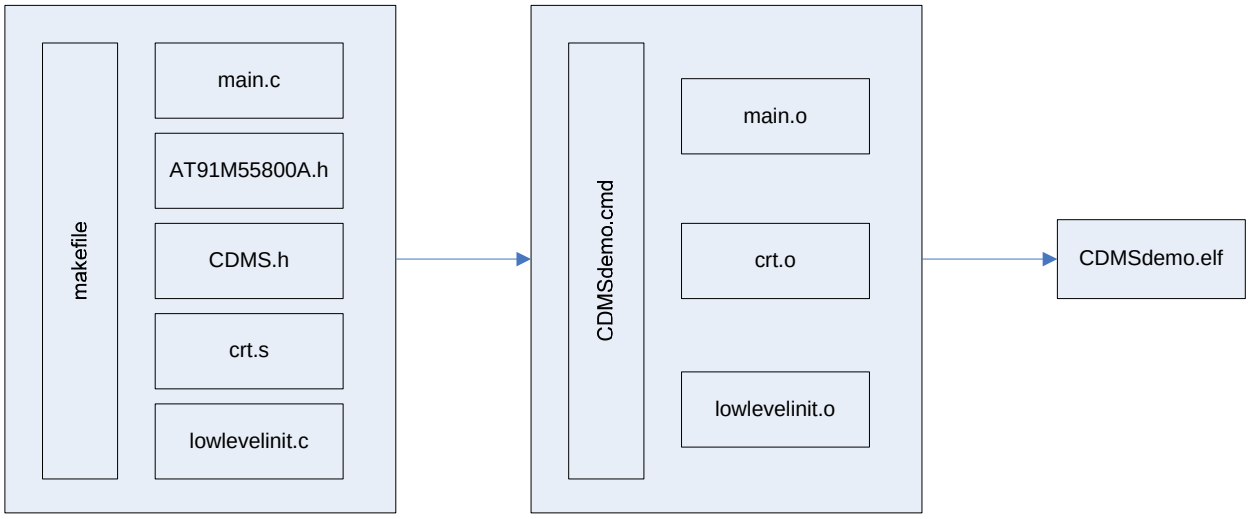


Figure 96 : Required files

The files required will be shortly presented on the following section:

- Main source
  - main.c

    This is the file where the C test function will be written in.
  - AT91M5880A.h

    This is the standard H file for the AT91M55800A microprocessor. ARM peripherals are memory-mapped, so all I/O registers are defined in this file so it is not required to type in the absolute memory addresses.
  - CDMS.h

    This is the definition file for the CDMS prototype board. It will define the external memories definition, the external bus interface initialization, the master clock and some other used I/O definitions.
  - crt.s

    This assembly startup file defines some symbols to set the various stack sizes and mode bits. This file sets up the interrupts vectors, the interrupts modes and stacks.
  - lowlevelinit.c

    This C language routine determines the low level initialization for the CDMS. This file initializes, for example, the watchdog, the clock sources, the remap command, and the default interrupts handler vectors.
- Compiler file
  - makefile

    A Make file is a text file referenced by the Make command. The makefile describes the target compilation; it contains information such as the dependences on the level source and dependences for the order of compilation.
- Linker file
  - CDMSdemo.cmd

    This file is the linker script which defines how the code and data emitted by the GNU C compiler and assembler are to be loaded into memory. Generally code goes into Flash and variables into RAM.
- Output file
  - CDMSdemo.elf

    The CDMSdemo.elf file supplies information necessary for the operating system to create a process image suitable for executing the code and accessing the data contained within the file.

    The executable and linking format (ELF) was originally developed by Unix System laboratories and is rapidly becoming the standard in file format. The ELF standard is growing in popularity because it has greater power and flexibility than the a.out and COFF binary format. Some of the capabilities of ELF are dynamic linking, dynamic loading, imposing runtime control on a program, and an improved method for creating shared libraries.

9.3 Uploading new software procedure

For reason of reliability the flight software will be written on a ROM memory. However, uploading new software should be interesting. In order to save new software, a non volatile memory chip will be added on the CDMS board. This non-volatile memory will contain a copy of the initial software which is programmed on the ROM. This section describes how it will be possible to boot from two different memory chips.

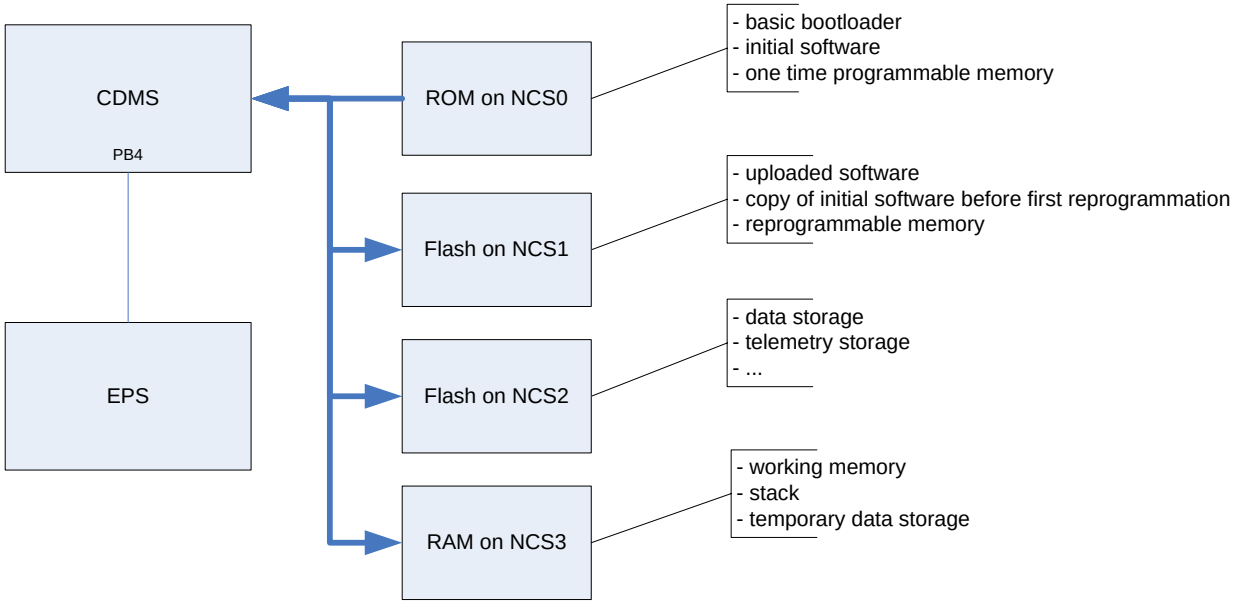


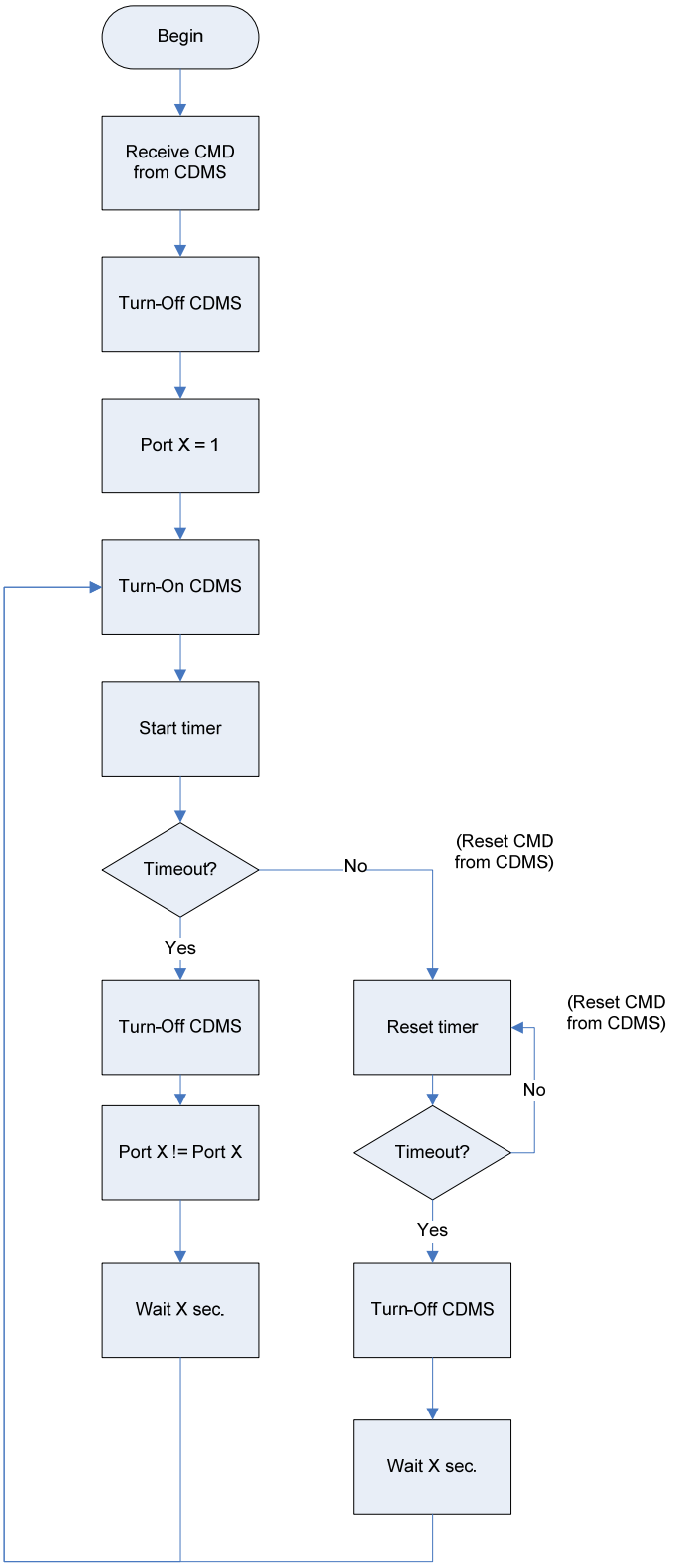
Figure 97 : Memory distribution

When the microcontroller is powered up, it will start reading from the ROM module at address 0x0000. This memory chip contains both the software needed to boot the system and all other applications that are running on the CDMS. The software in this memory is permanent and cannot be changed. The boot loader is split into two parts: a basic boot mode and an advanced boot. The basic boot is stored in the ROM and will set up the controller minimally by configuring only the most vital internal parameters. Minimizing the setup in the basic boot allows for more flexibility if it later is desired to change different parameters on the microcontroller.

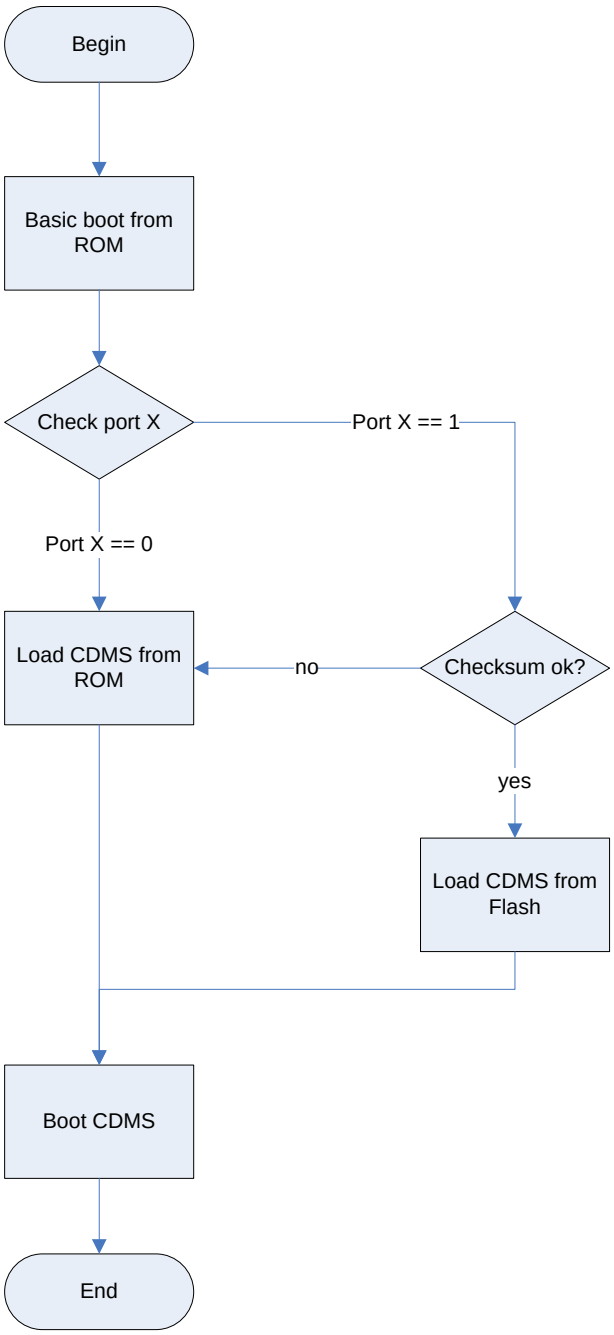
When the CDMS has completed the basic boot, it will check a dedicated pin (Port PB4, in this overview). If this port is high it indicates that the FLASH memory chip contains new software and will try to boot with the new software. Otherwise it will continue its boot from ROM.

The following figures will present the principle for booting either with FLASH memory or ROM.

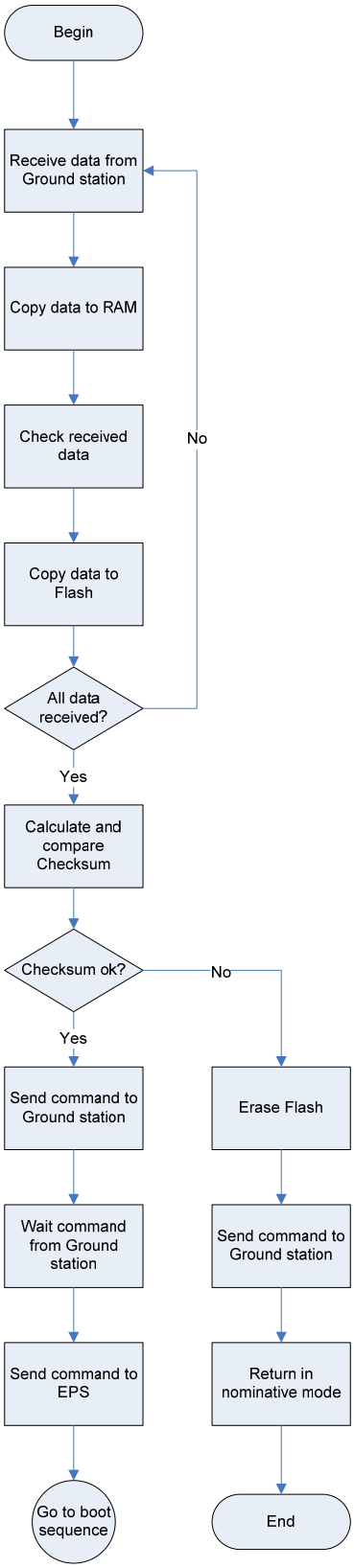
9.3.1 EPS boot sequence



9.3.2 CDMS boot sequence



9.3.3 Uploading new software



10 TEST

The main goal of the test is to verify that the CMD board filled all the requirements written in the specification. The various types of test and their goal are presented below.

- Functional test

  The goal of this test is to control that the CDMS board is able to ensure the tasks (measurements, data storage, and data processing) which are required. Those tasks must be able to work under all the conditions and constraints which the CDMS is judicious to support.

  The functional tests will take place before then, either during, or after the others tests defined below.

- Vibration test

  The goal of this test is to check that the CDMS board is compatible with the constraints of vibration of its environment (Launch, jettisoning, detumbling…).

  This test consists of a succession of random vibrations in three dimension, as well as shocks in six axes.

  A functional test must be carried out before and after the vibration test in order to check that the board is always able to work properly.

- Vacuum and thermal test

  The goal of this test is to check that the board is compatible with the temperatures of its environment, particularly during transport and into orbit. Moreover, it makes it possible to check that the outgazing is minimal in order not to disturb the optical part of the payload.

  This test let to eliminate a source from breakdown: infant mortality of the components.

- EMC/EMI/ESD/Radiation test

  Theses tests will verify the capacity of the CDMS to works, in a satisfactory way, in an environment subjected to electromagnetic (EM), electrostatic (ES) and radiation constraints.

  Functional test before and during the board is subjected to those various constraints must be performed.

- Benchmark

  The goal of this test is to check that the CDMS reaches the performances quantified in the specification. It makes it possible to define the degree of quality of the CDMS (board temperature, consumption, behavior with voltage variations…).

- Inspection (physical properties measurement, insulation, storage)

  The goal of this test is to inspect the CDMS board to control if the following points correspond to the specifications:

  - Weight
  - Total dimensions
  - Cleanliness
  - Welding
  - Fixing of the components

10.1 Test procedure

This section will present the procedure and information required to make a test. That information are useful in order to determine where a problem occurs and correct the test procedure in the case a test has been approved but the CDMS encounter a problem with the functionality corresponding to the test during the mission.

- Participants and responsibilities

  This part lists all the personnel mobilized to carry out the tests, like their function and responsibilities.

- Description of the tested board

  This section should briefly describe the CDMS board, in particular a list of its functionalities like data storage, acquisition of measurements of sensors, ADCS algorithm processing…

  The diagram of the functional architecture can also appear.

- Equipment used during the tests

  This part must contain the list of all the material equipment used to carry out the tests, including the model and serial number.

- Software used during the tests

  This part must contain the list of all the software used to perform the tests, including the version.

- Test configuration

  This part contains a list (test bench setting A, test bench setting B…) of possible configurations to set up before each test.

  Each Bench Setting is composed of a succession of handling to carry out to bring the board, the equipment and the software of test in a ready state for the test.

- Test conditions

  This part contains a list of environmental conditions that must be filled during the phase of the tests, such as:

  - Temperature
  - Humidity
  - Pressure
  - Cleanliness (dust)

- Particular precautions

  This part lists some important precautions to take to do the test, like:

  - Wearing an antistatic bracelet
  - Supplying the equipment of measure through an insulation transformer
  - Connecting the measurement probe with the CDMS powered off

All tests carried out are isolated on a "Test Procedure Sheet". An example applicable to the CDMS is presented below.

TEST EQUIPMENT LIST						
No.	Designation	Manufacturer	Type	Serial Number	Calibration validity	Remarks
EQ1	CDMS board	HEVs	V1.0		NA	-
TE2	Adjuste Power Supply 0V-24V	AS_CH	XXXXXX	XXXXXX	NA	-
TE3	Digital Oscilloscope	Tektronics	TDS-210	XXXXXX	NA	-
TE4	Probe 10x	---	---	XXXXXX	NA	-

TEST SOFTWARE LIST					
Designation	Plateform	Version	Site	Date	Remarks
Eclipse platform	PC windows XP	V3.2.0	HEVs	XXX	
OpenOCD	PC windows XP	2006re100	HEVs	XXX	JTAG connection
.elf executable test file	ARM7TDMI	V1.0	HEVs	XXX	

TEST PROCEDURE SHEET				
Name:	AD Conversion test			
Identification:	Functional test 1			
Objectives:	To check if an analog measured value is correctly converted and stored in RAM memory.			
Description:				
Step	**Set up and execution**			
	Hardware configuration			
FA1	Used test bench setting A			
	Software configuration			
	Using OpenOCD to transfer program in Flash memory			
	Execution			
	Execution & parameter verification	**Limits**	**Measured**	**Remark**
FA2	Power Off CDMS board and wait 10sec.			
FA3	Power on CDMS board			
FA4	Download the test program			
FA5	Apply a voltage of 0.000V Successively Vcc/2 then Vcc on the Analog Input 1 pin			
FA6	Run de test program for each value of voltage. Verify that the displayed function readed in the memory is : 0x00000000, 0x000001EF, 0x00000FFF	Value +/- 5%		
	Outcomes (files and reportf)		**Status**	**Date & Visa**

Figure 98 : CDMS test example

11 CONCLUSION

An embedded system according to the specification has been designed but not tested. The development and provisioning of the prototype board took much more time than expected; it has not been possible to do all that has been envisaged for this project, such as assembly and basic functional tests. A development environment has been deployed in order to write the test software for the CDMS and program the target. This environment is based on free GNU tools.

As the project is in constant evolution, the specifications will probably change in the future. But at that time the baseline of the Swisscube has been drawn.

The end of the phase B is planned for the end of February of 2007. At this time all subsystems groups will provide a prototype in order to envisage a first integration test.

12 FUTURE WORK

This first thing to do is the assembly of the printed card board then to do some tests which will proves that the hardware will work properly. The dimensioning of the lath-up protection must be done with the measured value of current consumption. After that it will be necessary to determine with the others subsystems that the prototype of the CDMS fills the desired functional specifications.

At that time a meticulous work to optimize the board must be done. The chapter 8.9 has presented the different analysis which will enable to determine which elements must be improved.

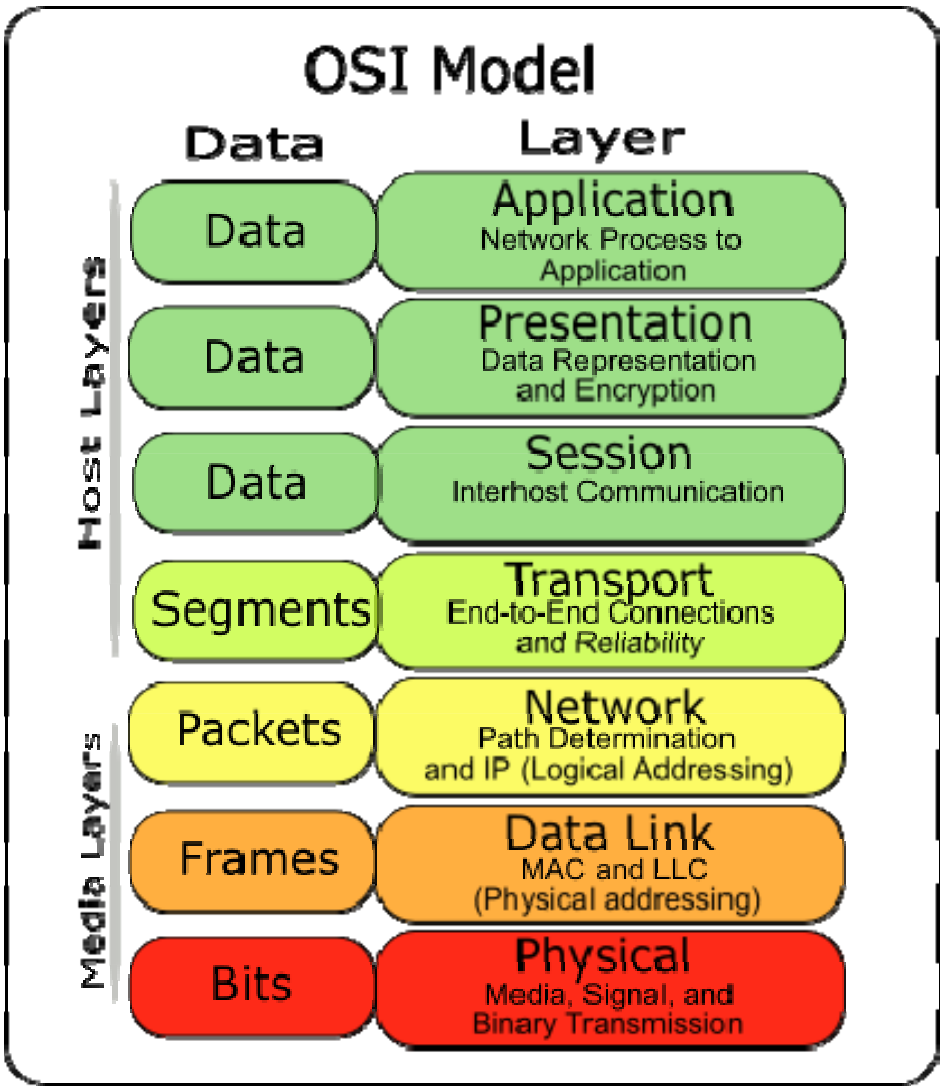
In a more remote future a second board based on the prototype will be manufactured in correlation with the results of the analysis and test.

23 November 2006

Tapparel Pierre-André

The OSI model divides the function of a protocol into a series of layers. Each layer has the property that it only uses the function of the layer below, and only exports functionality to the layer above. A system that implements protocol behaviour consisting of a series of these layers is known as a 'protocol stack'. Protocol stacks can be implemented either in hardware or software, or a mixture of both. Typically, only the lower layers are implemented in hardware, with the higher layers being implemented in software.

The OSI reference model is a hierarchical structure of seven layers that defines the requirements for communications between two systems.



A.1 Description of OSI layers

- Layer 7: Application Layer

The Application Layer is closest to the end user, provides a means for the user to access information on the network through an application. This layer is the main interface for the user to interact with the application and therefore the network. Some examples of application layer implementations include Telnet, File Transfer Protocol (FTP), and Simple Mail Transfer Protocol (SMTP).

- Layer 6: Presentation Layer

The Presentation Layer relieves the Application Layer of concern regarding syntactical differences in a message's data representation with the end-user systems. MIMI encoding, data compression, and similar manipulation of presentation is done at this layer to present the data as a service or protocol developers sees fit.

- Layer 5: Session Layer

The session layer provides the mechanism for managing the dialogue between end-user application processes. It provides for either duplex or half-duplex operation and establishes check pointing, adjournment termination, and restart procedures. The OSI model made this layer responsible for 'graceful close' of sessions, which is property of TCP (Transmission Control Protocol), and also for session check pointing and recovery, which is not usually used in the Internet protocol suite.

- Layer 4: Transport Layer

The Transport layer provides transparent transfer of data between en users, thus relieving the upper layers from any concern with providing reliable and cost- effective data transfer. The transport layer controls the reliability of a given link. Some protocols are state and connection orientated. This means that the transport layer can keep track of the packets and retransmit those that fail. The best known example of a layer 4 protocol is TCP. It is the layer that converts message into TCP (Transmission Control Protocol) or UDP (User Datagram Protocol) packets.

- Layer 3: Network Layer

The Network layer provides the functional and procedural means of transferring variable length data sequence from a source to a destination via one or more networks while maintaining the quality of service requested by the Transport layer. The Network layer performs routing, flow control, segmentation/documentation, and error control functions. The best known example of layer 3 protocol is the Internet Protocol (IP).

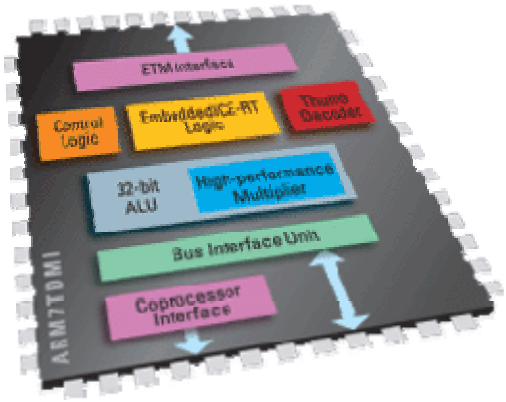
- Layer 2: Data Link Layer

The Data Link layer provides the functional and procedural means to transfer data between network entities and to detect and possibly correct errors that may occur in Physical Layer

- Layer 1: Physical Layer

The Physical layer defines all the electrical and physical specifications for devices. This includes the layout of pins, voltages, and cable specification. The major functions and services performed by the physical layer are.

This appendix provides an introduction to the ARM7TDMI processor embedded in the AT91 series of microcontroller. This section will identify the basic components of the processor architecture and also take a detailed look at the processor functions and the instruction sets.



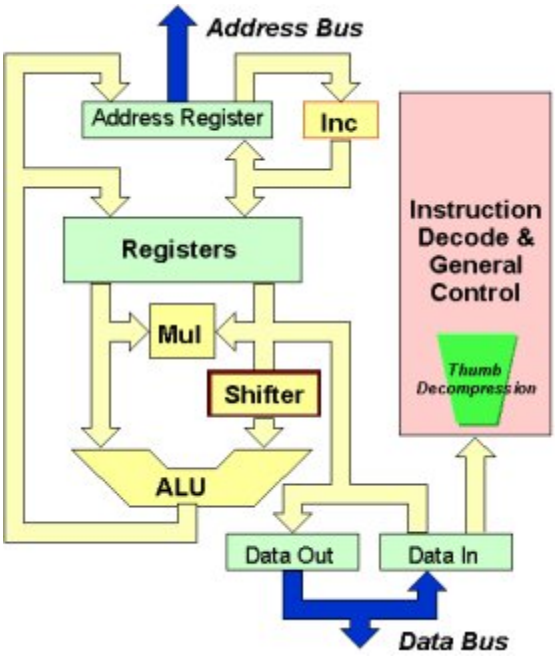
The ARM7TDMI processor is a member of the Advanced RISC machine family of general purpose 32-bit microprocessor, which offers high performance for very low power consumption and price.

ARM7TDMI means:

- ARM7 : 32-bit Advanced RISC Machine
- T : Thumb architecture extension.
- D : Debug extension
- M : Enhanced multiplier
- I : Embedded ICE macrocell extension

The following Figure will present the block diagram of the ARM7TDMI processor.

- Von Neumann Architecture
- 32-bit Data Bus
- 32-bit Address Bus
- Three stage pipeline (Fetch, Decode, Execute)
- 37 32-bit registers
- 32-bit ARM instruction set
- 16-bit THUMB instruction set
- 32*8 Multiplier Barrel Shifter



The processor architecture is Von Neumann load and store architecture which is characterized by a single data and address bus for instructions and data.

The ARM7TDMI is a 3-stage pipeline 32-bit RISC processor. Pipelining is employed so that all parts of the processing and memory systems can operate continuously. Typically, while one instruction is being executed, its successor is being decoded, and a third instruction is being fetched from memory. The ARM7TDMI processor is built around a bank of 37 32-bit registers including six status registers.

Its outstanding feature is the 16-bit THUMB subset of the most commonly used 32-bit instructions. This gives 16-bit code density coupled with 32-bit processor performance. It features an integrated 32x8 multiplier and 32-bit barrel shifter. Five independent internal buses allow a high degree of parallelism in instruction execution.

A.1 ARM7TDMI operating states and modes

The ARM7TDMI processor has two operating states. The ARM state which executes 32-bit, word aligned ARM instructions and the THUMB state which can execute 16-bit, halfword aligned THUMB instructions. From the programmer's point of view, the processor can be in one of these two states.

- Entering THUMB state: BX instruction with the state bit (bit 0) set in the operand register. Automatically on return from an exception (IRQ, FIQ, ABORT, SWI …), if the exception was entered with the processor in THUMB state.

- Entering ARM state: BX instruction with the state bit clear in the operand register. Automatically on the processor taking an exception. In this case, the Program Counter (PC) is placed in the execution mode's link register.

This processor supports seven modes of operation:

- User (usr) : The normal ARM program execution state.
- FIQ (fiq) : Designed to support a data transfer or channel process.
- IRQ (irq) : Used for general-purpose interrupt handling.
- Supervisor (svc) : Protected mode for the operating system.
- Abort mode (abt) : Entered after a data or instruction prefetch abort.
- System (sys) : A privileged user mode for the operating system.
- Undefined (und) : Entered when an undefined instruction is executed.

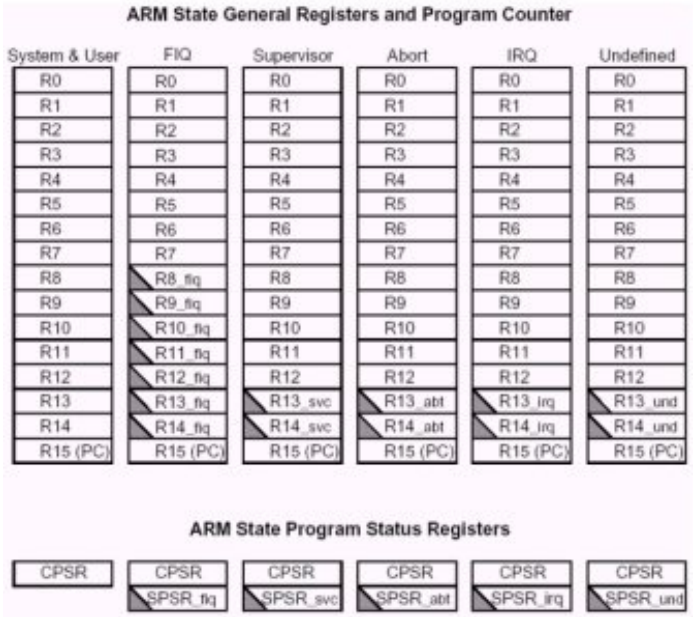
Mode changes may be made under software control, or may be brought about external interrupts or exception processing. Most application programs will execute in User mode. The non-user modes, known as privileged mode, are entered in order to service interrupts or exception, or to access protected resources.

A.2 ARM7TDMI registers

The ARM7TDMI has a total of 37 registers: 31 general-purpose 32-bit registers and 6 status registers. These registers cannot all be seen at once. The processor state and operating mode dictate which registers are available to the programmer.

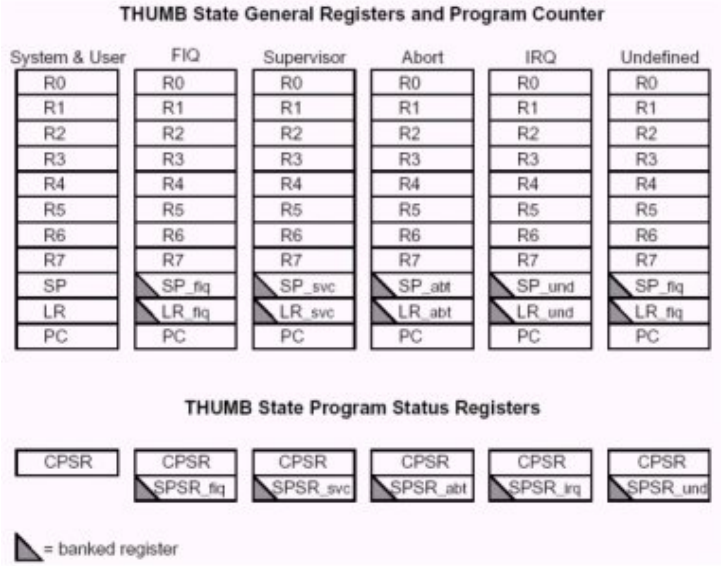
A.2.1 ARM state register set

In ARM state, 16 general registers and one or two status registers are visible at any one time. In privileged (non-user) modes, mode-specific banked registers are switched in. The registers R0 to R15 are directly accessible. All of these except R15 are general purpose, and may be used to hold either data or address values. In addition to these, there is a seventeenth register used to store status information. The following figure shows which registers are available in each mode. The banked registers are marked with a shaded triangle.



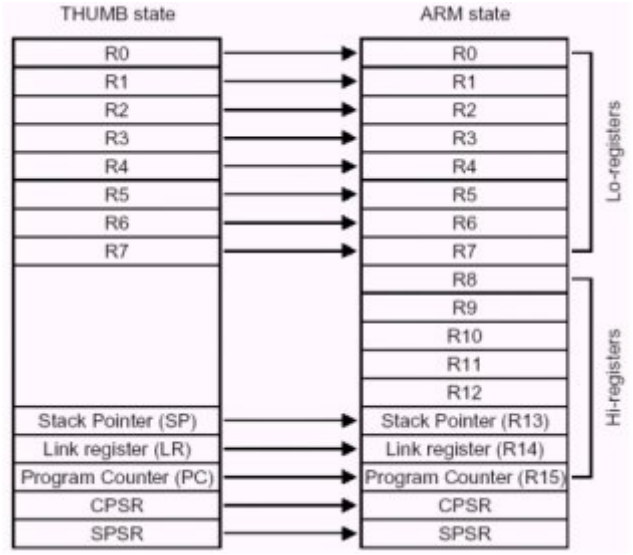
A.2.2 THUMB state register set

The THUMB state register set is a subnet of the ARM state set. The programmer has direct access to eight general registers, R0 to R7, as well as the Program Counter (PC), a stack register (SP). A link register (LR), and the CPSR. There are banked Stack Pointers, Link Registers and Saved Process Status Registers (SPSR) for each privileged mode.



A.2.3 ARM and THUMB registers relationship

The THUMB state registers relate to the ARM state register in the following way: THUMB state Ro to R7 and ARM state R0 to R7 are identical, THUMB state CPSR and SPSR and ARM state CPSR and SPSR are identical, THUMB state SP maps onto ARM state R13. THUMB state LR maps onto ARM state R14 and the THUMB state Program Counter maps onto the ARM state Program Counter (R15).



A.2.4 Program Status Register

The ARM7TDMI contains a Current Program Status Register (CPSR), plus five Saved Program Status Registers (SPSR) used for exception handlers. The different functions of this register are to hold information about the most recently performed ALU operation, to control the enabling and disabling of interrupts and to set the processor operating mode.

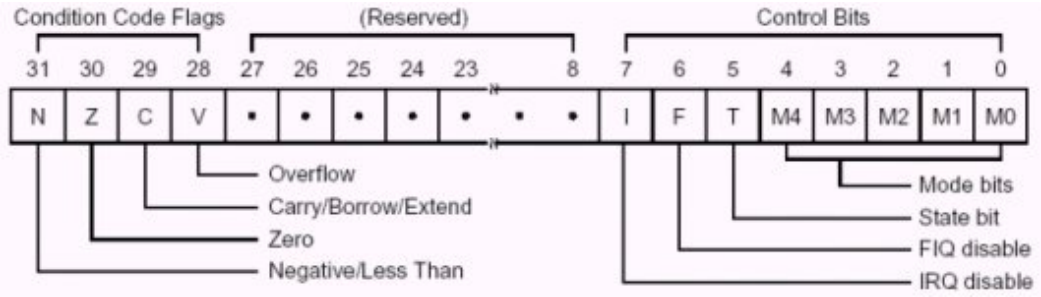


Figure 1 : Program status register

The CPSR is accessible in all processor modes. It contains condition code flags, interrupt disable bits, the current processor mode and other status and control information. Each exception mode also has a SPSR that is used to preserve the value of the CPSR when the associated exception occurs.

- Condition Code Flags

  The N, Z, C and V bits may be changed as a result of arithmetic and logic operations, and may be tested to determine whether an instruction should be executed. In ARM state, all instructions may be executed conditionally. In THUMB state, only the Branch instruction is capable of conditional execution.

- Control bits

  The I, F, T and M[4:0] will be changed when an exception arises. If the processor is operating in a privileged mode, they can also be manipulated by software.

  - The T bit reflects the operating state. When this bit is set, the processor is executing in THUMB state, otherwise it is executing in ARM state. This is reflected on the TBIT external signal. Note that the software must never change the state of the TBIT in the CPSR. If this happens, the processor will enter an unpredictable state.
  - The I and F bits are the interrupt disable bits. When set, these disable the IRQ and FIQ interrupts respectively.
  - The M4, M3, M2, M1 and M0 bits (M[4:0]) are the mode bits. These determine the processor's operating mode. Not all combinations of the mode bits define a valid processor mode. Only those explicitly described shall be used. The user should be aware that if any illegal value is programmed into the mode bits, M[4:0], then the processor will enter an unrecoverable state. If this occurs, reset should be applied.

A.3 Exceptions

Exceptions arise whenever the normal flow of a program has to be halted temporarily, for example to service an interrupt from a peripheral. The ARM supports seven types of exception and has a privileged processor mode for each type of exception.

- The following figure presents the different ARM exception vectors.

Address	Exception	Mode in Entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software Interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	Reserved	Reserved
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

- When handling an exception, the ARM7TDMI:
  - Preserves the address of the next instruction in the appropriate Link Register.
  - Copies the CPSR into the appropriate SPSR.
  - Forces the CPSR mode bits to a value which depends on the exception.
  - Forces the PC to fetch the next instruction from the relevant exception vector.
  - It may also set the interrupt disable flags to prevent otherwise unmanageable nesting of exceptions.
  - If the processor is in THUMB state when an exception occurs, it will automatically switch into ARM state when the PC is loaded with the exception vector address.
- On completion, the exception handler:
  - Moves the Link Register, minus an offset where appropriate, to the PC.
  - Copies the SPSR back to the CPSR
  - Clears the interrupt disable flags, if they were set on entry.

A.3.1 Exception sources

- Reset

  When the nRESET signal goes 'LOW', the ARM7TDMI abandons the executing instructions and then continue to fetch instructions from incrementing word addresses. When nRESET goes 'HIGH' again, the processor forces M[4:0] to 10011 (supervisor mode), set the I and F bits in the CPSR, and clears the CPSR's T bit. Then, it forces the PC to fetch the next instruction from address 0x00.

  - CPSR <= Supervisor + I + F
  - PC <= 0x00000000

- Undefined instruction

  When ARM7TDMI comes across an instruction which it cannot handle, it takes the undefined instruction trap. After emulating the failed instruction, the trap handler should execute the following irrespective of the state (ARM or THUMB): MOVS PC, R14_und. This restores the CPSR and returns to the instruction following the undefined instruction.

o	LR_und	<=	Undefined instruction address + #4
o	PC	<=	0x00000004
o	CPSR	<=	Undefined + I
o	Return with	:	MOVS pc, lr

- Prefetch abort

  Prefetch abort occurs during an instruction prefetch. If a prefetch abort occurs, the prefetched instruction is marked as invalid, but the exception will not be taken until the instruction reaches the head of the pipeline.

o	LR_abt	<=	Aborted instruction address + #4
o	SPSR_abt	<=	CPSR
o	PC	<=	0x0000000C
o	CPSR	<=	Abort + I
o	Return with	:	SUBS pc, lr, #4

- Data abort

  Data abort occurs during a data access. If a data abort occurs, the action taken depends on the instruction type.

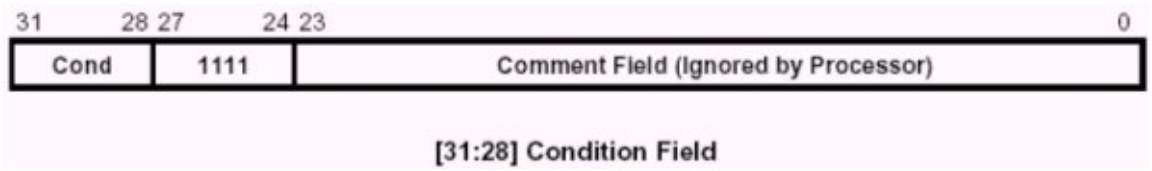
o	LR_abt	<=	Aborted instruction + #8
o	SPSR_abt	<=	CPSR
o	PC	<=	0x0000010
o	CPSR	<=	Abort + I
o	Return with	:	SUBS pc, lr, #4 or SUBS pc, lr, #8

- Software interrupt

  The software interrupt instruction (SWI) is used for entering Supervisor mode, usually to request a particular supervisor function. A SWI handler should return by executing the following irrespective of the state (ARM or THUMB): MOV PC, R14_svc. This restores the PC and CPSR, and return to the instruction following the SWI.

o	LR_svc	<=	SWI address + #4
o	SPSR_svc	<=	Supervisor + I
o	Return with	:	MOV pc, lr

The bottom 24-bits of the instruction are ignored by the processor, and may be used to communicate information to the supervisor code.



- Interrupt request

  The IRQ (interrupt request) exception is a normal interrupt caused by a 'LOW' level on the nIRQ input. It may be disabled at any time by setting the I bit in the CPSR, though this can only be done from a privileged (non-User) mode. Irrespective of whether the exception was entered from ARM or THUMB state, an IRQ handler should return from the interrupt by executing SUBS PC, R14_irq, #4.

o	LR_irq	<=	P - #4
o	SPSR_irq	<=	CPSR
o	PC	<=	0x00000018
o	CPSR	<=	Interrupt + I
o	Return with	:	SUBS pc, lr, #4

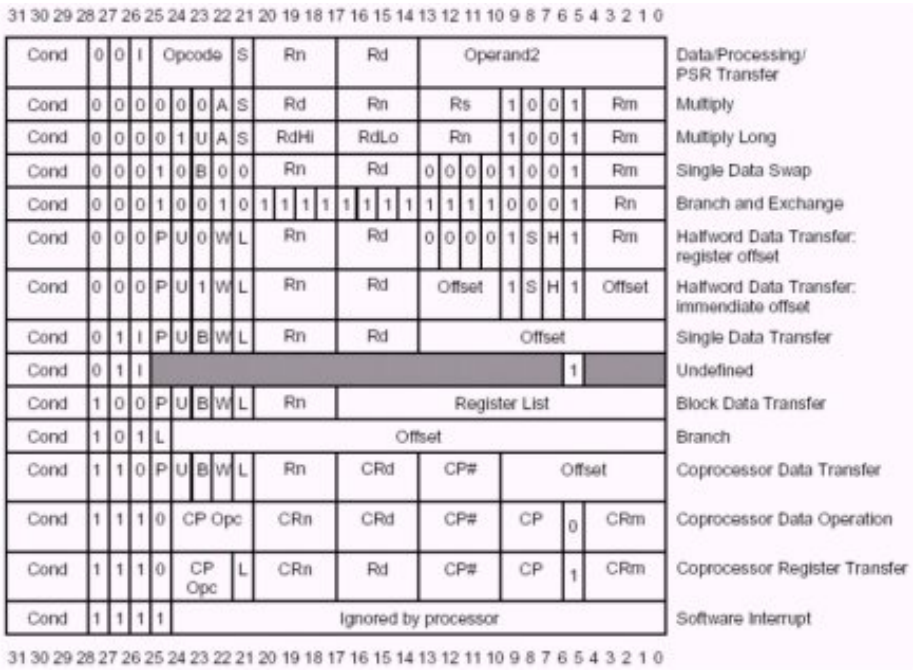
- Fast interrupt request

  The FIQ exception is designed to support a data transfer or channel process. FIQ is externally generated by taking the nFIQ input 'LOW'. Irrespective of whether the exception was entered from ARM or THUMB state, a FIQ handler should leave the interrupt by using SUBS PC, R14_fiq, #4. FIQ may be disabled by setting the CPSR's F flag.

o	LR_fiq	<=	PC - #4
o	SPSR_fiq	<=	CPSR
o	PC	<=	0x0000001C
o	CPSR	<=	fast interrupt + I + F
o	Return with	<=	SUBS pc, lr, #4

A.4 ARM instruction set

The following figure presents an overview of ARM instruction set.



A.4.1 The Condition Field

All ARM instructions an be conditionally executed, which means that their execution may or may not take place depending on the values of the N, Z, C and V flags in the CPSR. Every instruction contains a 4-bit condition code field in bits 31 to 28. This field determines the circumstances under which an instruction is to be executed.

There are fifteen different conditions, each represented by a two-character suffix that can be appended to the instruction's mnemonic.

Code	Suffix	Flags	Meaning
0000	EQ	Z set	Equal
0001	NE	Z clear	Not equal
0010	CS	C set	Unsigned higher or same
0011	CC	C clear	Unsigned lower
0100	MI	N set	Negative
0101	PL	N clear	Positive or zero
0110	VS	V set	Overflow
0111	VC	V clear	No overflow
1000	HI	C set and Z clear	Unsigned higher
1001	LS	C clear or Z set	Unsigned lower or same
1010	GE	N equals V	Greater or equal
1011	LT	N not equal to V	Less than
1100	GT	Z clear AND (N equals V)	Greater than
1101	LE	Z set OR (N not equal to V)	Less than or equal
1110	AL	(ignored)	always

A.4.2 Branch Instruction

All ARM Processors support a branch instruction that allows a conditional branch forwards or backwards up to 32Mbytes. As the Program Counter (PC) is one of the general purpose registers (R15), a branch or jump can also be generated by writing a value to this register R15.

A subroutine call is a variant of the standard branch, the Branch with Link instructions preserves the address of the instruction after the branch (the return address) in register 14 (link register or LR).

A load instruction provides a way to branch anywhere in the 4Gbyte address space. A 32-bit value is loaded directly from memory into the PC, causing a branch

The ARM7TDMI processor that support the THUMB instruction set also support a branch instruction (BX) that jumps to a given address, and optionally switches executing THUMB instructions.

List of branch instructions:

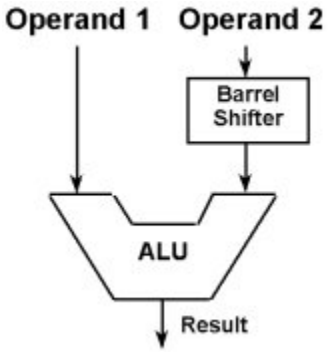
- B Branch
- BL Branch with link
- BX Branch and exchange instruction set

Example:

```
        B       label       ; branch unconditionally to label
        BCC     label       ; branch to label if carry flag is clear
        BEQ     label       ; branch to label if zero flag is set

        MOV     PC, #0      ; R15=0, branch to location zero

        BL      func        ; subroutine call function

func    MOV     PC, LR      ; R15=R14, return to instruction after the BL
        MOV     LR, PC      ; store the instruction‘s address after the next one into R14
        LDR     PC, = func  ; load a 32-bit value into the program counter
```

A.4.3 Data Processing

ARM has 16 data processing instructions. Most data processing instructions take two source operands (Move and Move Not have only one operand) and store a result in a register (except for the Compare and Test instructions which only update the condition codes). Of the two operands, one is always a register; the other is called a shift operand, and is either a immediate value or a register. If the second operand is a register value, it may have a shift applied to it before it is used as the operand to the ALU.



The following figure presents the list of data processing instructions.

Assembler Mnemonic	OP Code	Action
AND	0000	Operand1 AND operand2
EOR	0001	Operand1 EOR operand2
WUB	0010	Operand1 – operand2
RSB	0011	Operand2 operand1
ADD	0100	Operand1 + operand2
ADC	0101	Operand1 + operand2 + carry
SBC	0110	Operand1 – operand2 + carry –1
RSC	0111	Operand2 – operand1 + carry –1
TST	1000	As AND, but results is not written
TEQ	1001	As EOR, but result is not written
CMP	1010	As SUB, but result is not written
CMN	1011	As ADD, but result is not written
ORR	1100	Operand1 OR operand2
MOV	1101	Operand2 (operand1 is ignored)
BIC	1110	Operand1 AND NOT operand2 (Bit clear)
MVN	1111	NOT operand2 (operand1 is ignored)

A.4.4 Multiply instructions

Arm has two classes of multiply instruction. Normal (32-bit result) and Long (64-bit result). All multiply instructions take two register operands as the input to the multiplier. ARM does not directly support a multiply by constant instruction due to the efficiency of shift and add, or shift and reverse subtract instructions.

There are two multiply instructions that produce 32-bit results

- MUL, multiplies the values of two registers together, truncates the result to 32 bits, and stores the result in a third register.

- MLA, multiplies the value of two registers together, adds the value of a third register, truncates the result to 32 bits, and store the result into a fourth register (multiply and accumulate).

Example:

```
MUL     R4, R2, R1      ; set R4 to value of R2 multiplied by R1
MULS    R4, R2, R1      ; R4 = R2*R1, set N and Z flags.
MLA     R7, R8, R9, R3  ; R7 = R8*R9 + R3
```

There are four multiply instructions that produce 64-bit results (long multiply)

- Two of the variant multiply the value of two registers together and store the 64-bit result in a third and fourth register. There are a signed (SMULL) and unsigned (UMULL) variants.
- The remaining two variants multiply the values of two registers together, add the 64-bit value from a third and fourth register and store the 64-bit result back into those registers (third and fourth). There are also signed (SMLAL) and unsigned (UMLAL) variants. These instructions perform a long multiply and accumulate.

Example:

```
SMULL   R4, R8, R2, R3  ; R4 = bits 0 to 31; R8 = bits 32 to 63 of R2*R3
UMULL   R6, R8, R0, R1  ; R6, R8 = R0*R1
UMLAL   R5, R8, R0, R1  ; R5, R8 = R0*R1 + R5, R8
```

A.4.5 Load and Store instructions

Load and store instruction come in three types:

- Load and Store single register
  - Load register instructions can load a 32-bit word, a 16-bit halfword or an 8-bit byte from memory into a register.
  - Store register instructions can store a 32-bit word, a 16-bit halfword or an 8-bit byte from a register to memory.
  - List of load and store single register

```
LDR/STR     ; Load/Store word
LDRB/STRB   ; Load/Store byte
LDRH/STRH   ; Load/Store unsigned halfword
LDRSB       ; Load signed byte
LDRSH       ; Load signed halfword
```

- Load and Store multiple registers
  - Load and Store multiple instructions perform a block transfer of any number of the general purpose registers to or from memory
  - Four addressing modes are provided: pre-increment, post-increment, pre-decrement, post-decrement.

- List of load and store multiple instructions

LDM	; Load multiple
STM	; Store multiple

- Swap a register value with the value of a memory location
  - Swap can load a value from a register specified memory location, store the contents of a register to the same memory location, then write the loaded value to a register
  - List of Swap instructions

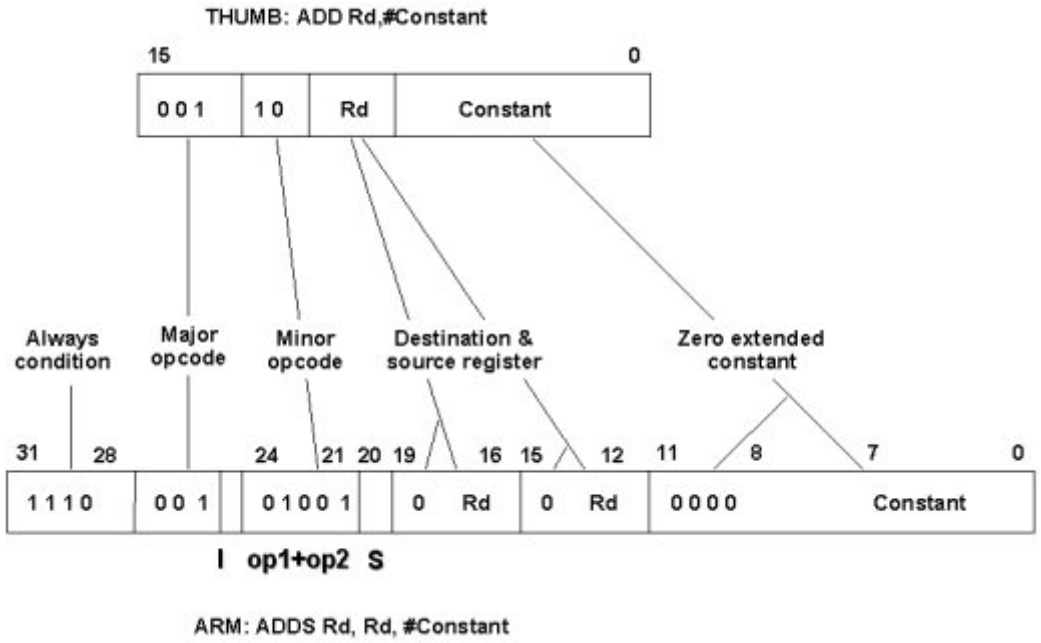
SWP	; Swap
SWPB	; Swap byte

A.5 THUMB instruction set

The following figure presents an overview of THUMB instruction set.

The Thumb instruction set is a subnet of the ARM instruction set, optimized for code density. Almost every Thumb instructions have an ARM instructions equivalent. The inline expansion of THUMB instruction to ARM instruction provides a Real time decompression. The THUMB instructions are not actually executed on the core. The core needs to know whether it is reading THUMB instructions or ARM instructions. It has two execution states ARM and THUMB, but did not have a mixed 16 and 32 bit instruction set.

The following figure presents the THUMB instruction set decompression.



A.5.1 Branch Instruction

THUMB supports four types of branch instruction. An unconditional branch that allows a forward or backward branch of up to 2kbytes; a conditional branch to allow forward and backward branch of up to 256bytes; a branch with link is supported with a pair of instructions that allow forward and backwards branches of up to 4Mbytes and a branch exchange instructions branches to an address in a register and optionally switches to ARM code execution.

List of branch instructions:

- B Conditional branch
- B Unconditional branch
- BL Branch with link
- BX Branch and exchange instruction set

A.5.2 Data processing instruction

THUMB data processing instructions are a subnet of the ARM data processing instructions. All THUMB data processing instructions set the condition codes.

List of data processing instructions:

- ADC Add with carry
- ADD Add
- AND Logical AND
- ASR Arithmetic shift right
- BIC Bit clear
- CMN Compare negative
- CMP Compare
- EOR Exclusive OR
- LSL Logical shift left
- LSR Logical shift right
- MOV Move
- MUL Multiply
- MVN Move Not
- NEG Negate
- ORR Logical OR
- ROR Rotate Right
- SBC Subtract with carry
- SUB Subtract
- TST Test

A.5.3 Load and Store instructions

Load and store instruction come in two types:

- Load and Store register instruction:

  Thumb supports eight types of load and store registers instructions.

  List of load and store registers instructions:

```
LDR     ; Load word
LDRB    ; Load unsigned byte
LDRH    ; Load unsigned halfword
LDRSB   ; Load signed byte
LDRSH   ; Load signed halfword
STR     ; Store word
STRB    ; Store byte
STRH    ; Store halfword
```

- Load and Store multiple instructions:

  THUMB supports four types of load and store multiple instructions. Two (a load and a store) are designed to support block copy. The other two instructions (called PUSH and POP) implement a full descending stack, and the stack pointer is used as the base register.

  List of load and store multiple instructions:

```
LDM     ; Load multiple
POP     ; Pop multiple
PUSH    ; Push multiple
STM     ; Store multiple
```

A.6 ARM vs. THUMB instruction set

In a theoretical way, the THUMB instruction set allows to implement code in twice less memory. It is however not the case in practice, because all the instructions of ARM instruction set inevitably do not have an equivalent in THUMB instruction set; these instructions must thus be carried out using several reducing THUMB instructions reducing as much the profit of memory. ARM speaks about 30% of place gained on average (in the place of the theoretical 50%) what is remarkable all the same.

It is interesting to note that under certain quite precise conditions, saving memory space is perhaps equivalent to a speed profit. For example, on a system with a external 16-bit data bus, each ARM instructions require two read cycles whereas the THUMB instruction requires only one. In that way a smaller code in memory can be faster.

A.6.1 ARM instruction set advantage

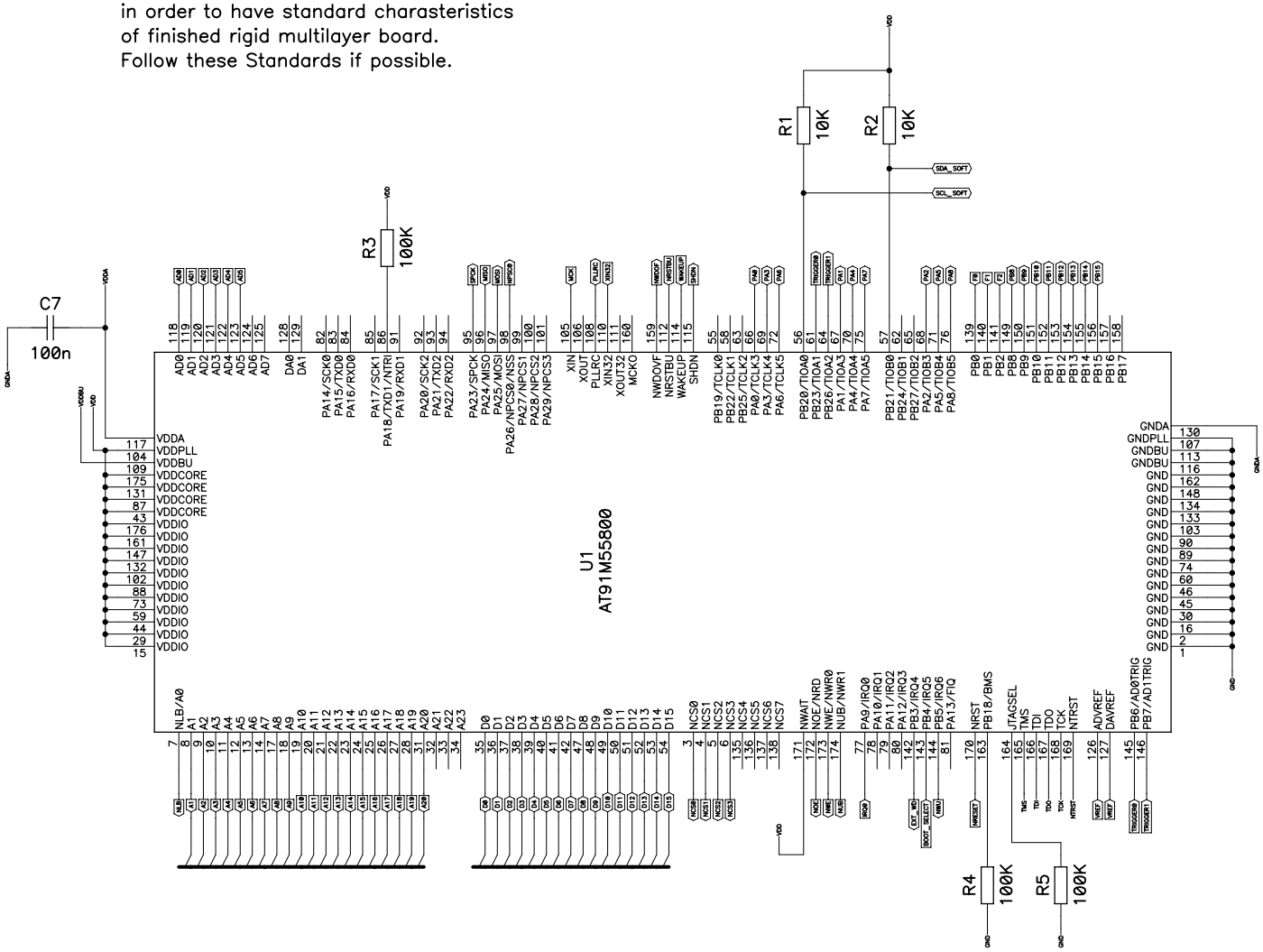
- All instructions are 32-bit long
- Most instructions are executed in one single cycle.
- Every instruction can be conditionally executed.

A.6.2 ARM instruction set advantage

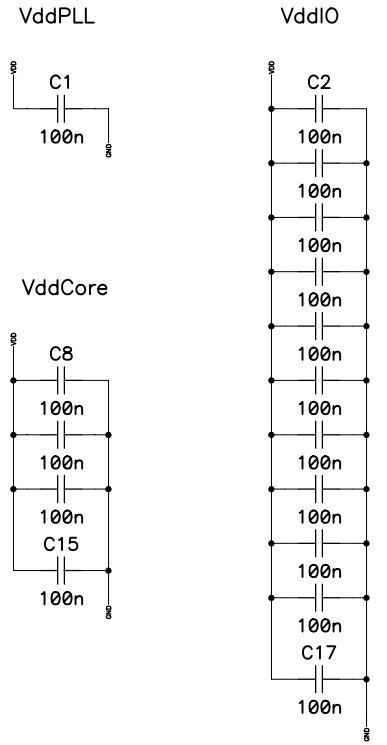
- All instructions are exactly 16 bits long to improve code density over other 32-bit architectures.
- The THUMB architecture stills use a 32-bit core.

CPU

See ECSS–Q–70–11A pages 37 to 39
in order to have standard charasteristics
of finished rigid multilayer board.
Follow these Standards if possible.



Decoupling

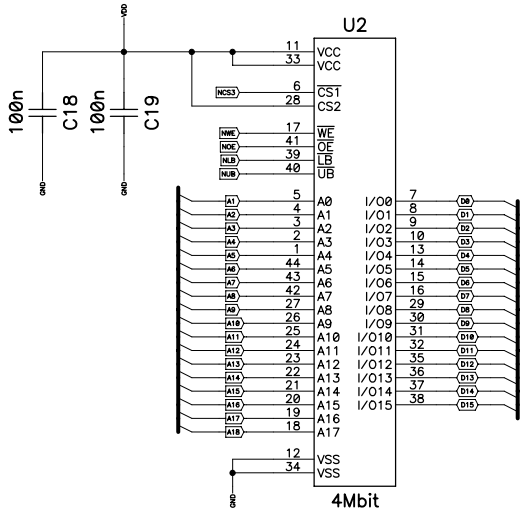


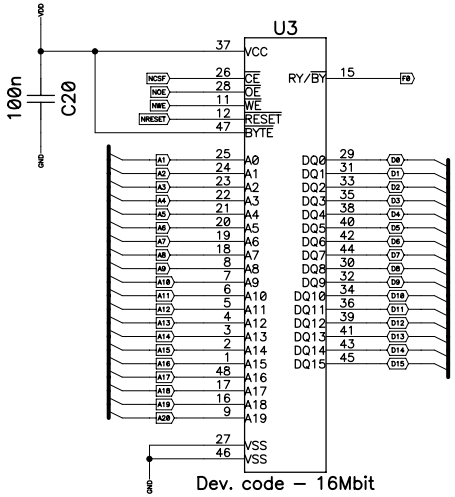
These decoupling capacitor must
be placed as near as possible
to corresponding pin.

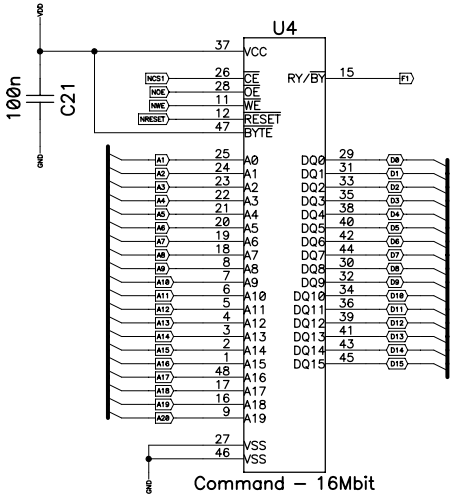
Vddbu decoupling capacitor (C39)
is presented on page 5.

CDMS – Swisscube Prototype Microcontroller	DES	25.09.2006 Tapparel Pierre–Andre
	REV	V1.2
HAUTE ECOLE VALAISANNE	1/6	{Path} CDMS_ 1.0f.sch

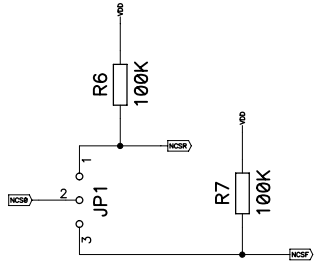
Memory



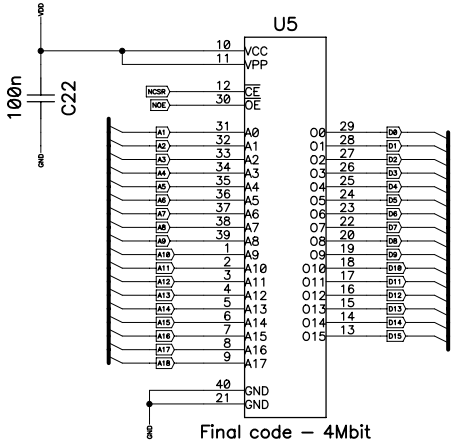


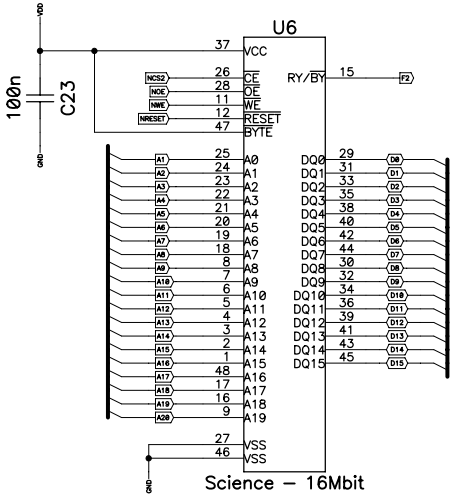


CS0 – selection jumper



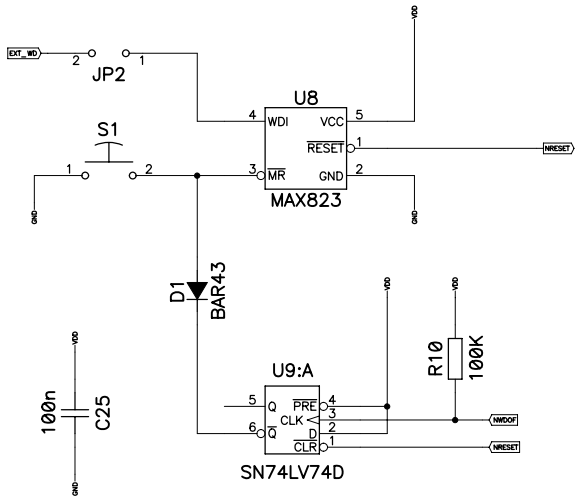
R6 and R7 are Pull–up resistor for respectively CSR and CSF signals.



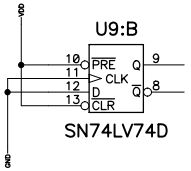


CDMS – Swisscube Prototype Memory	DES	25.09.2006 Tapparel Pierre–Andre
	REV	V1.2
HAUTE ECOLE VALAISANNE	2/6	{Path} CDMS_ 1.0f.sch

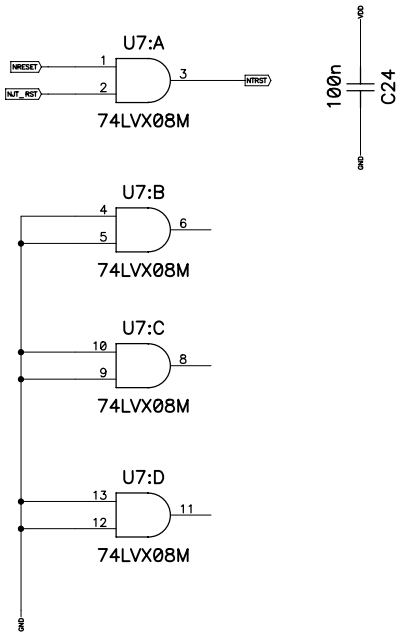
Reset switch



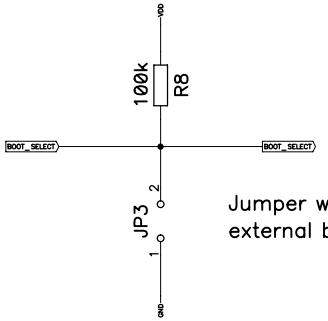
The IC MAX823 is not decoupled by a capacitor to detect small power fault. 74LV74 is decoupled.



JTAG reset

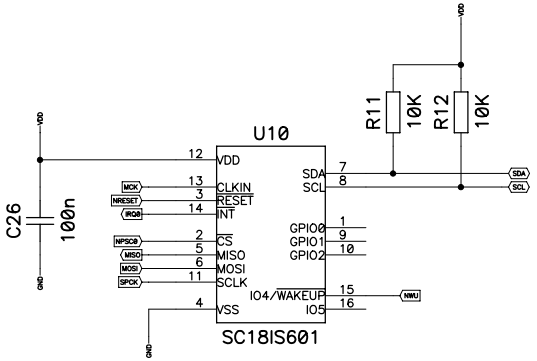


Boot selection jumper



Jumper which simulate external boot_ select signal

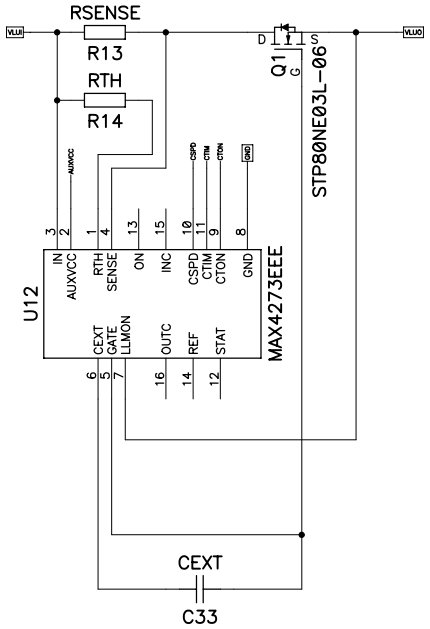
SPI to I2C bridge

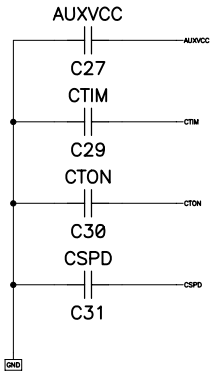


SW power-down: 0x30 0x5A 0xA5
NWU High: Power-down state
NWU Low: Exit power-down

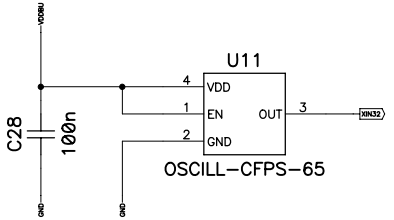
CDMS – Swisscube Prototype Divers	DES	25.09.2006 Tapparel Pierre-Andre
	REV	V1.2
HAUTE ECOLE VALAISANNE	3/6	{Path} CDMS_ 1.0f.sch

Latch-up protection

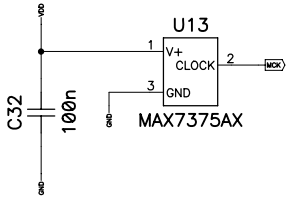




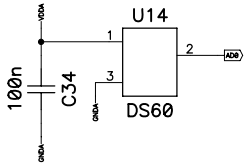
Oscillator – 32.768KHz



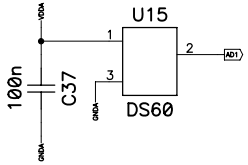
Oscillator – 4MHz



Temperature sensor

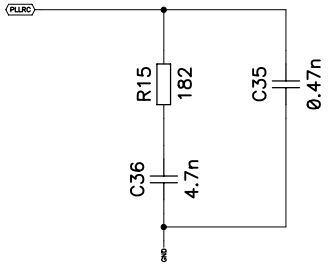


Nearest to microcontroller as possible



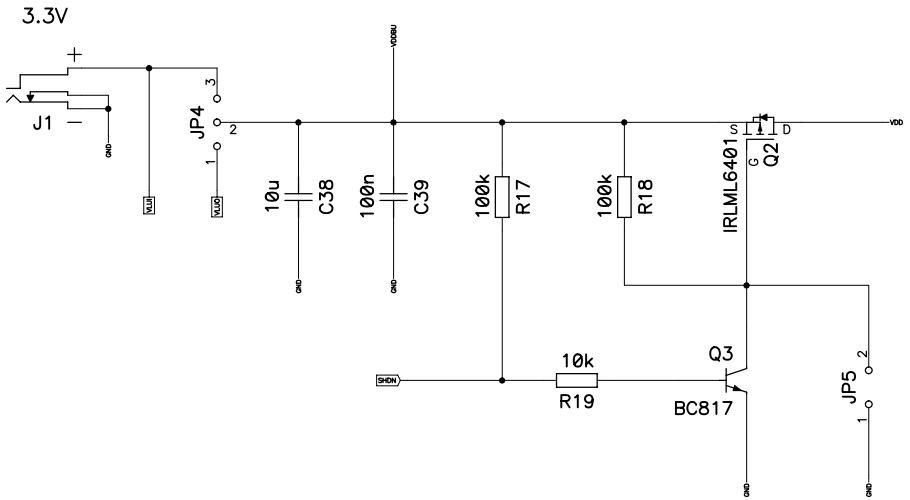
Near the memory chip

PLL – 4MHz to 20Mhz @ 3.3V



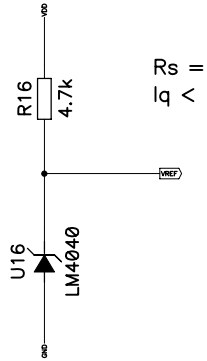
CDMS – Swisscube Prototype	LUP – Clock	DES	25.09.2006 Tapparel Pierre-Andre
		REV	V1.2
HAUTE ECOLE VALAISANNE		4/6	{Path} CDMS_ 1.0f.sch

Back up alimentation



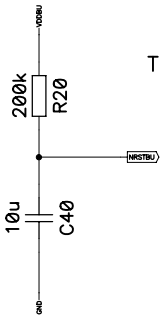
open: enable shutdown feature
closed: disable shutdown feature

2.5V – Vref



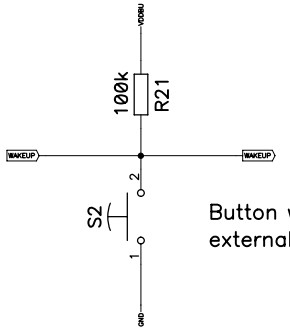
Rs = (Vs–Vr)/(Il+Iq)
Iq < 15mA

300ms min. reset pulse



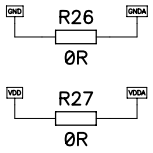
T = RxC (63.8%) = 2s

External Wakeup



Button wich simulate
external wakeup signal

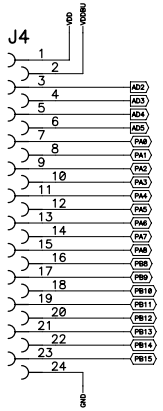
VddA – GNDA connection



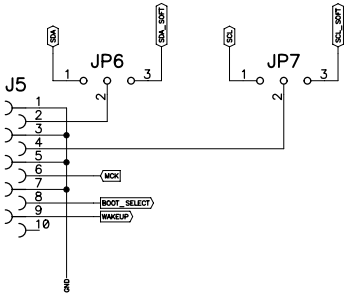
GNDA and VDDA should be connected
with one point to respectively GND and Vdd

CDMS – Swisscube Prototype Back–up		DES	25.09.2006 Tapparel Pierre–Andre
		REV	V1.2
HAUTE ECOLE VALAISANNE		5/6	{Path} CDMS_ 1.0f.sch

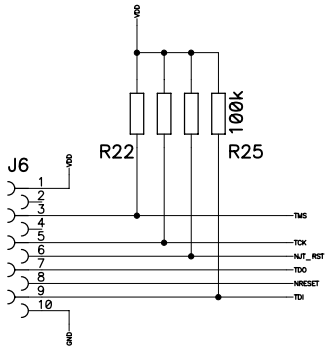
External connector



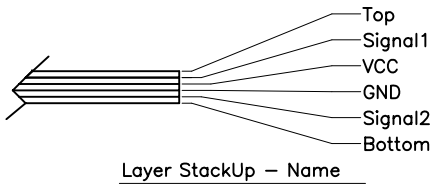
Subsystem connector



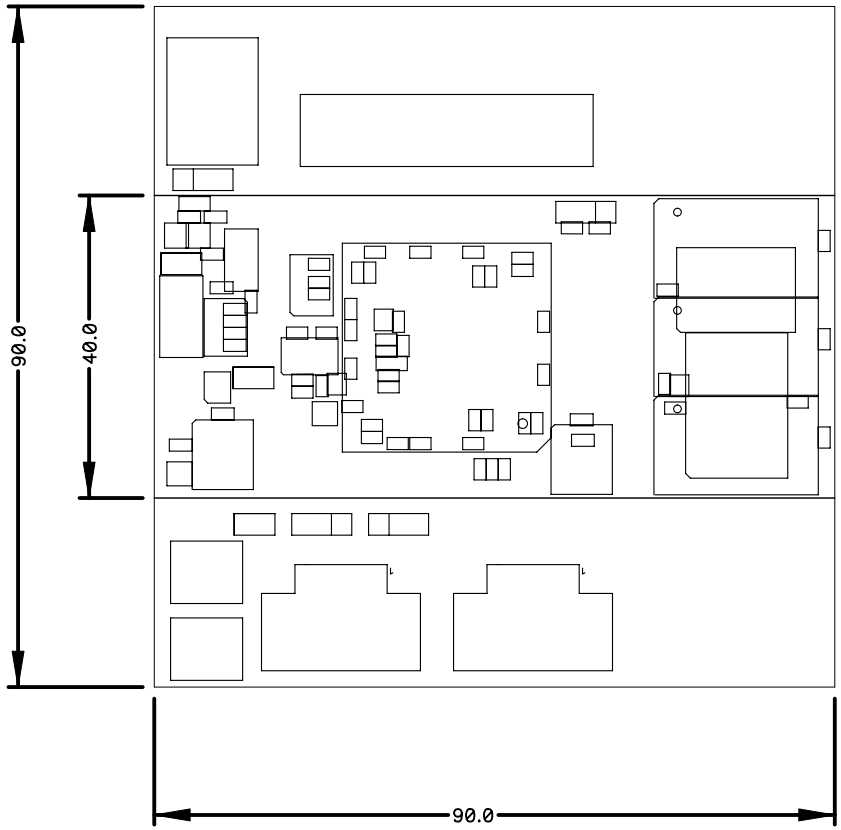
JTAG Connector



CDMS – Swisscube Prototype Connector	DES	25.09.2006 Tapparel Pierre-Andre
	REV	V1.2
HAUTE ECOLE VALAISANNE	6/6	{Path} CDMS_ 1.0f.sch

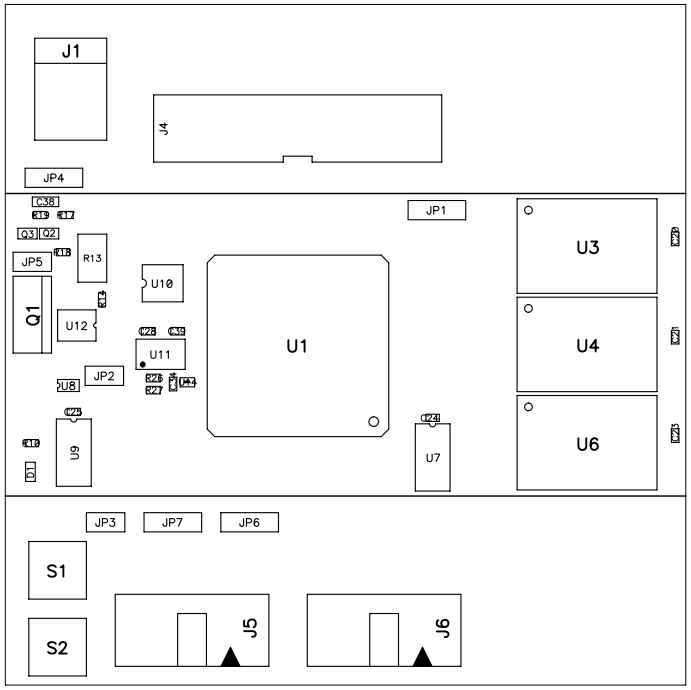


Layer StackUp – Name

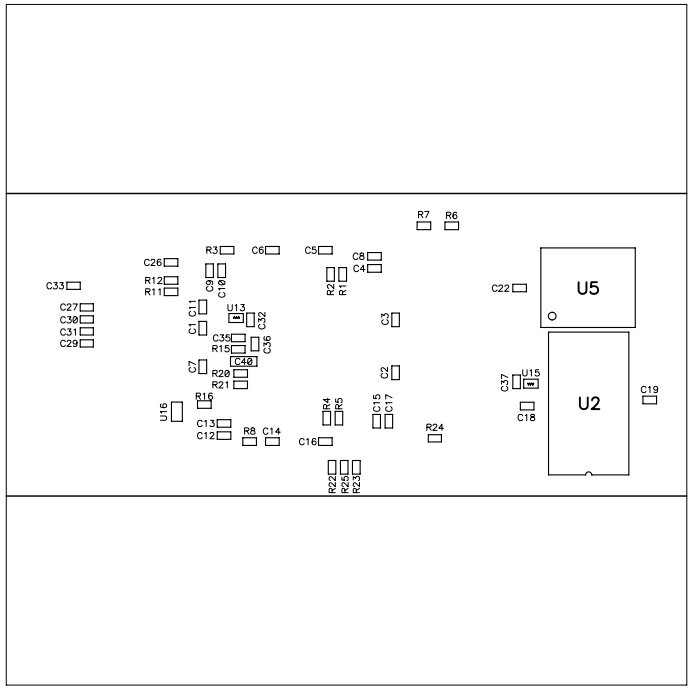


Drill Table			
Hole Dia (mm)	Symbol	Quantity	Plated
0.150	+	348	Yes
0.800	X	11	Yes
0.900	Y	44	Yes
1.000	⊤	18	Yes
2.600	⧖	2	Yes
3.200	⋈	1	Yes

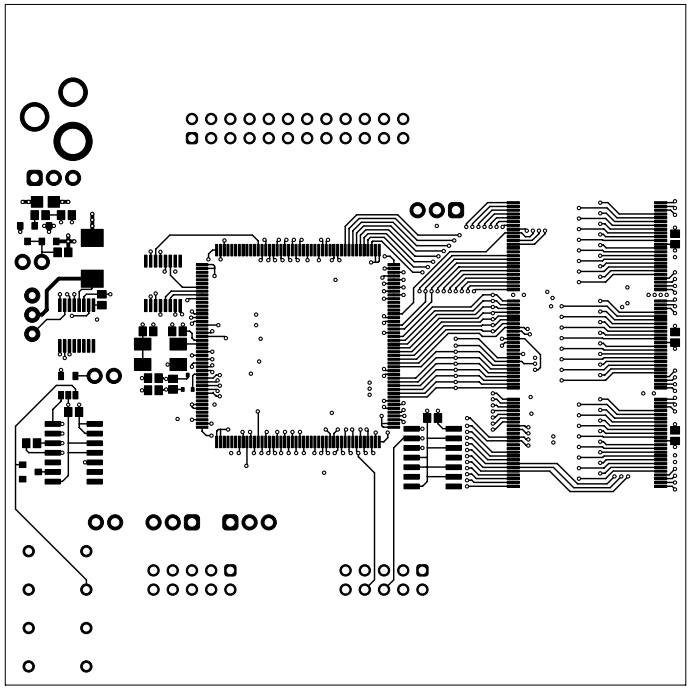
Implantation	DES	Date
	REV	V1.0
HAUTE ECOLE VALAISANNE	PCB/ CDMS_ 1.2_ 2002.pcb	



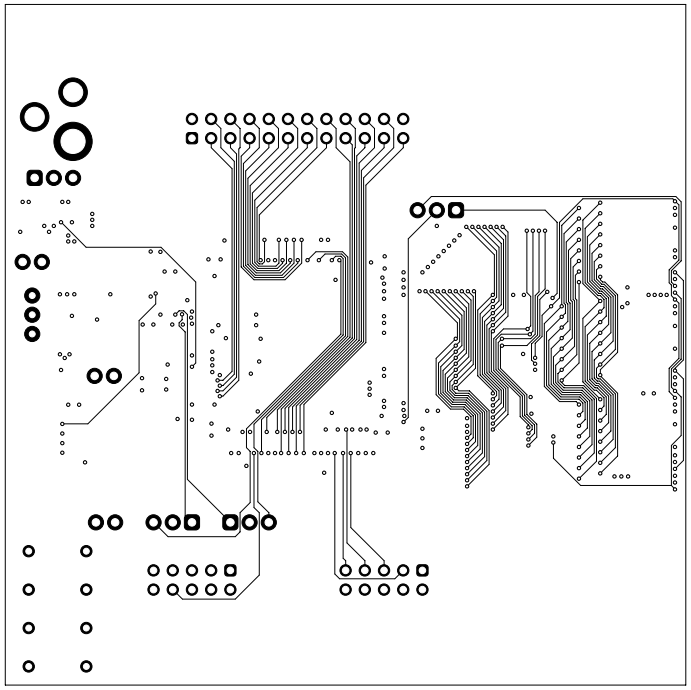
Top Implantation	DES	Date
	REV	V1.0
HAUTE ECOLE VALAISANNE	PCB/ CDMS_ 1.2_ 2002.pcb	



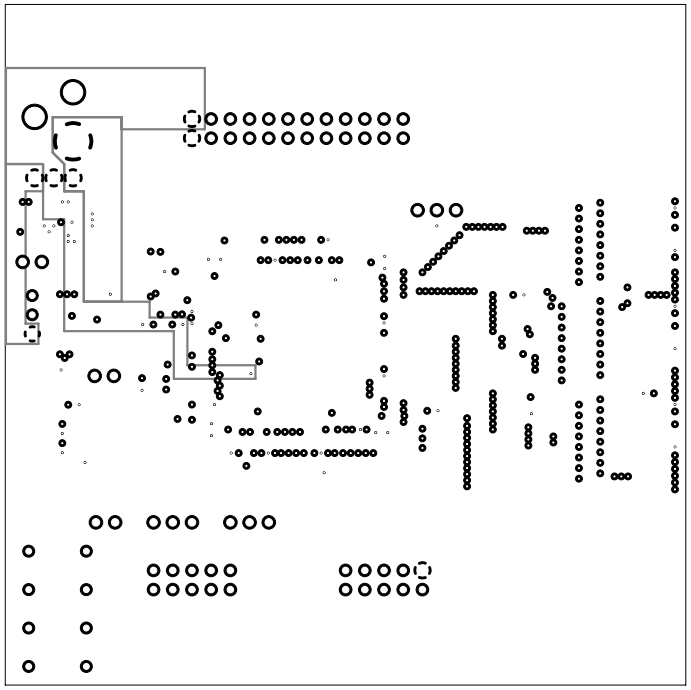
Bottom Implantation	DES	Date	
	REV	V1.0	
HAUTE ECOLE VALAISANNE	PCB/ CDMS_1.2_2002.pcb		



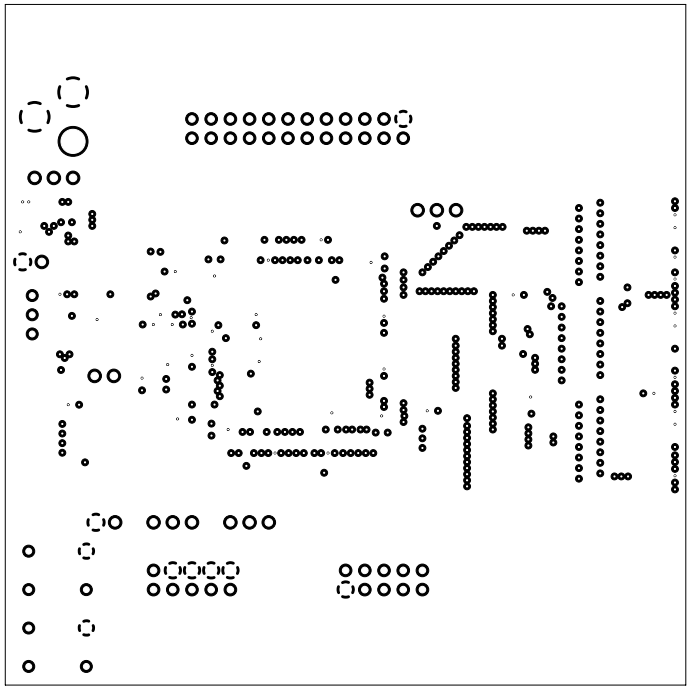
Top layer	DES	Date	
	REV	V1.0	
HAUTE ECOLE VALAISANNE	PCB/ CDMS_ 1.2_ 2002.pcb		



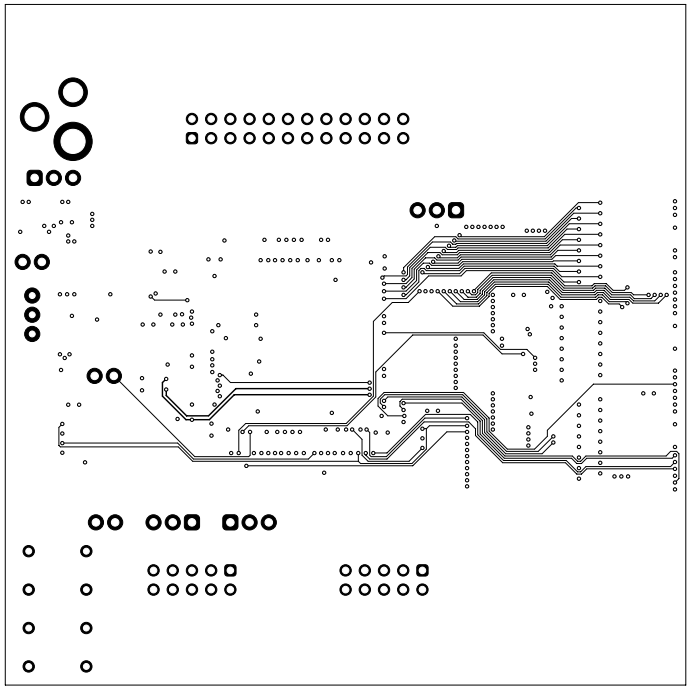
Signal 1 layer	DES	Date	
	REV	V1.0	
HAUTE ECOLE VALAISANNE	PCB/ CDMS_1.2_2002.pcb		



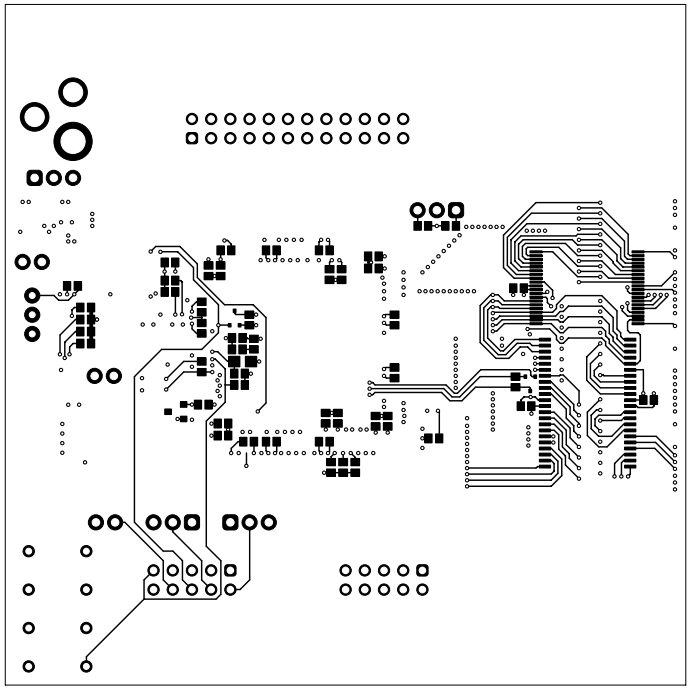
Vcc layer	DES	Date	
	REV	V1.0	
HAUTE ECOLE VALAISANNE	PCB/ CDMS_ 1.2_ 2002.pcb		



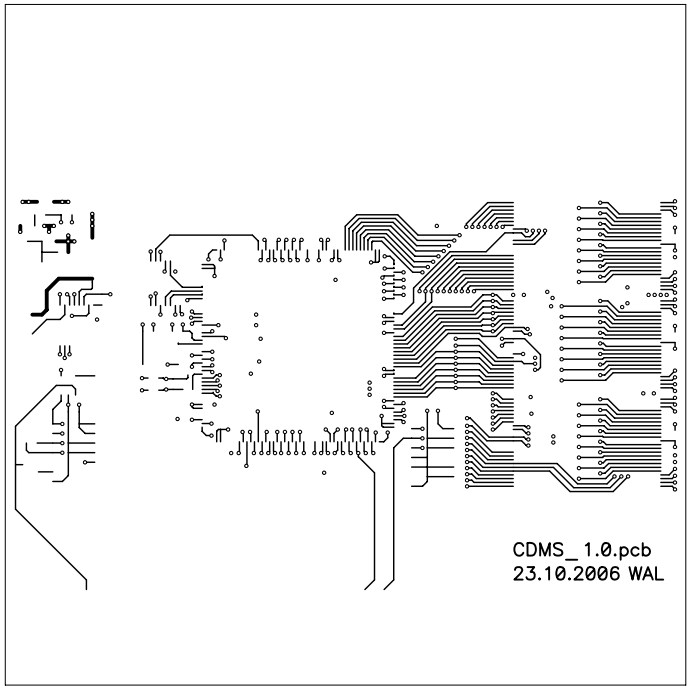
GND layer	DES	Date	
	REV	V1.0	
HAUTE ECOLE VALAISANNE	PCB/ CDMS_ 1.2_ 2002.pcb		



Signal 2 layer	DES	Date	
	REV	V1.0	
HAUTE ECOLE VALAISANNE	PCB/ CDMS_ 1.2_ 2002.pcb		

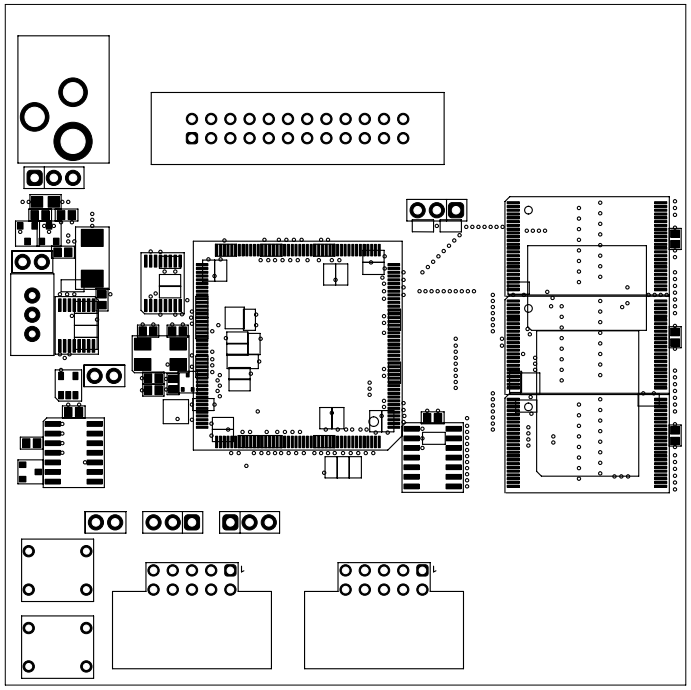


Bottom layer	DES	Date	
	REV	V1.0	
HAUTE ECOLE VALAISANNE	PCB/ CDMS_ 1.2_ 2002.pcb		



Drill Table			
Hole Dia (mm)	Symbol	Quantity	Plated
0.150	+	348	Yes
0.800	X	11	Yes
0.900	Y	44	Yes
1.000	⊤	18	Yes
2.600	⧖	2	Yes
3.200	⋈	1	Yes

Vias	DES	Date
	REV	V1.0
HAUTE ECOLE VALAISANNE	PCB/ CDMS_ 1.2_ 2002.pcb	



Drill Table			
Hole Dia (mm)	Symbol	Quantity	Plated
0.150	+	348	Yes
0.800	X	11	Yes
0.900	Y	44	Yes
1.000	⊤	18	Yes
2.600	⧖	2	Yes
3.200	⋈	1	Yes

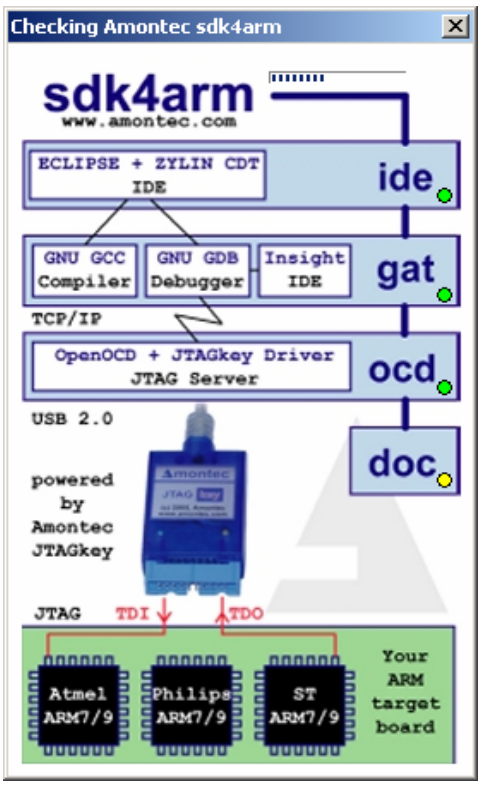
Drill	DES	Date
	REV	V1.0
HAUTE ECOLE VALAISANNE	PCB/ CDMS_ 1.2_ 2002.pcb	

This appendix will make a small introduction on how to use Eclipse, from the compilation to the debug using OpenOCD.

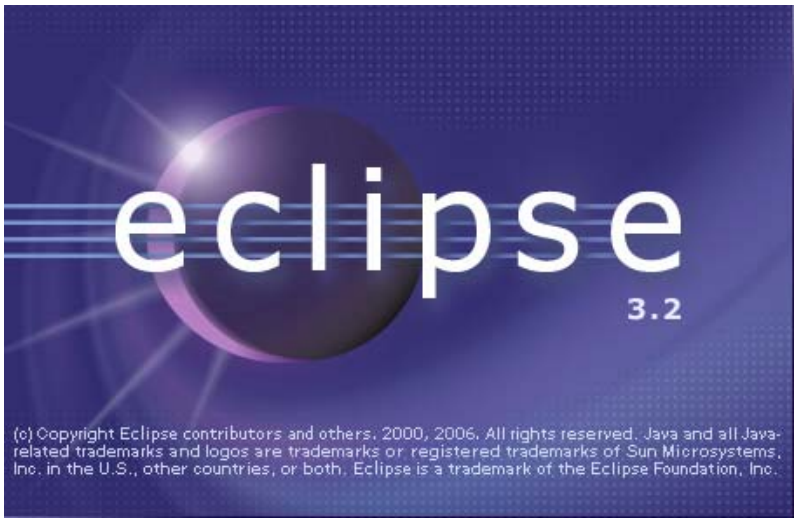
A.1 Eclipse new project

Inside the directory where the Amontec adk4arm was installed, double click on the file "sdk4arm Eclipse.exe" to start Eclipse. Or simply double click on the desktop shortcut.

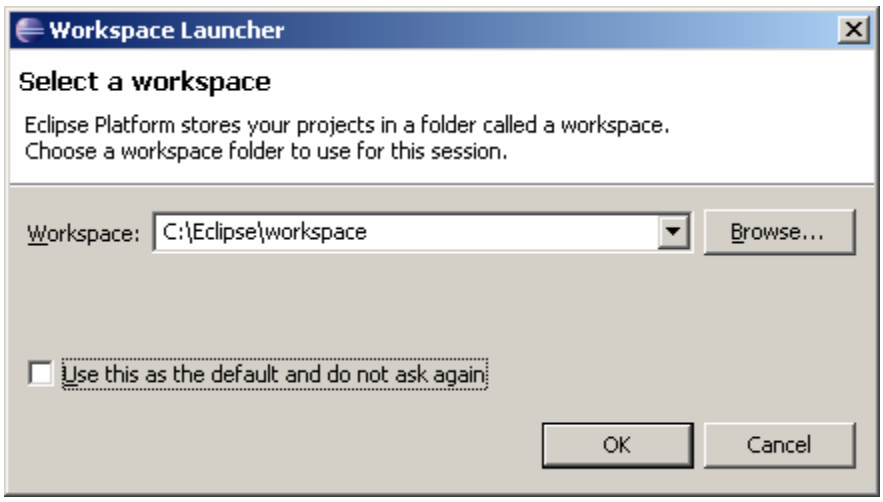
During the start of Eclipse the following screen appears showing the loading advance.



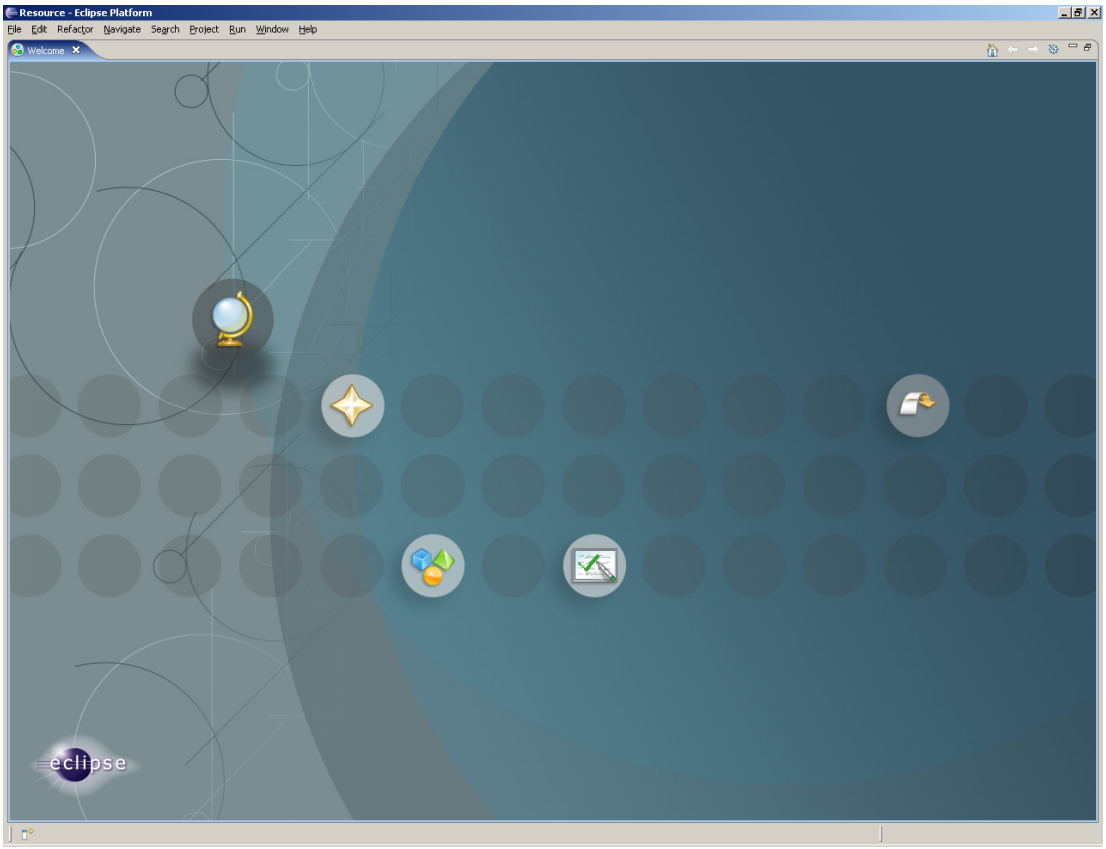
After the start of Eclipse the following splash screen is presented:



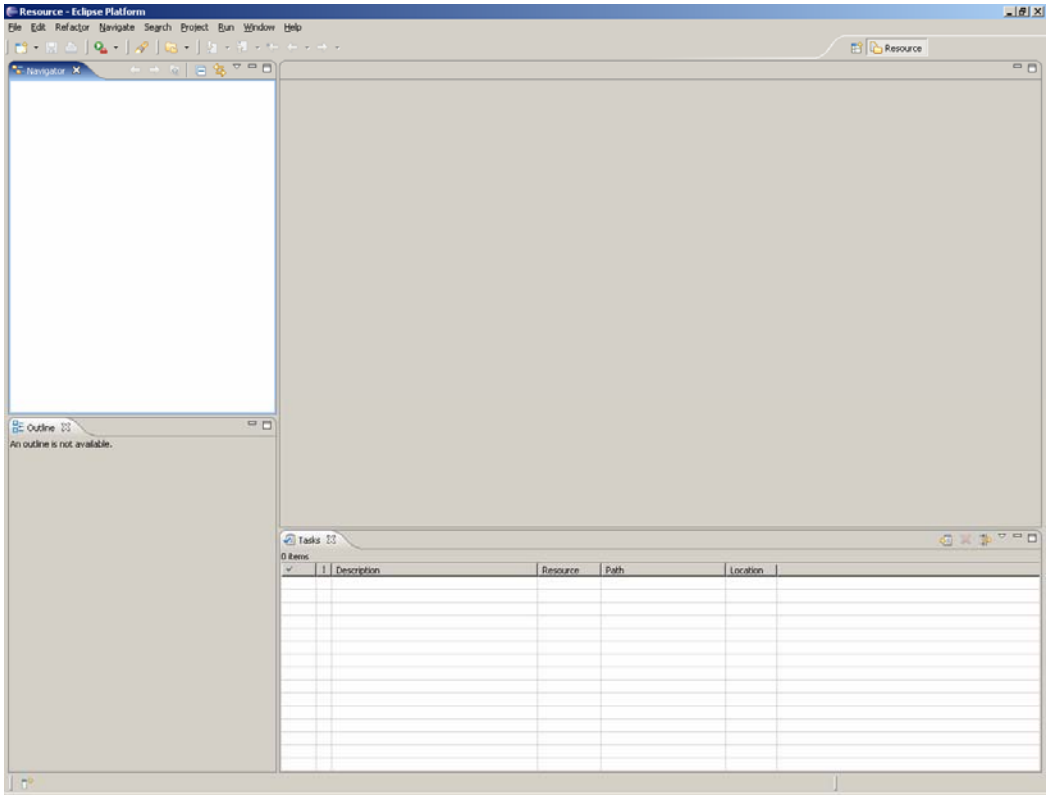
If not, you have a problem with the Java Runtime Environment. After a while Eclipse want to know where to store your projects:



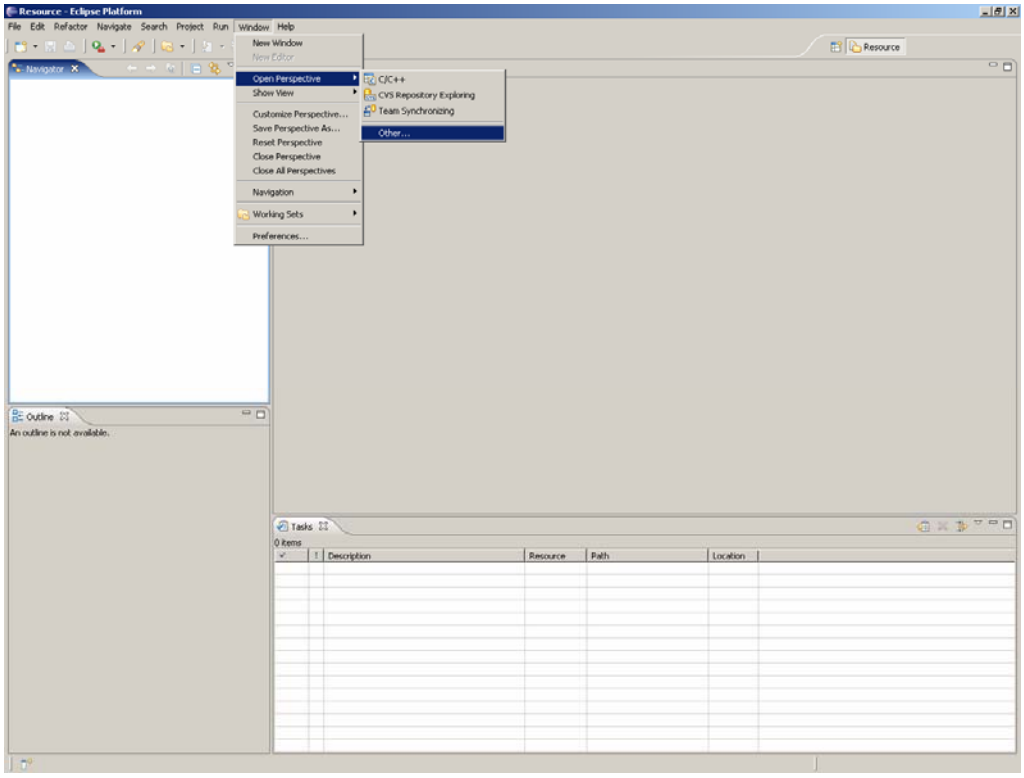
For the Eclipse workspace the folder C:\eclipse\workspace will be used. Enter the directory and press “OK”. The following "Welcome" window will appear:



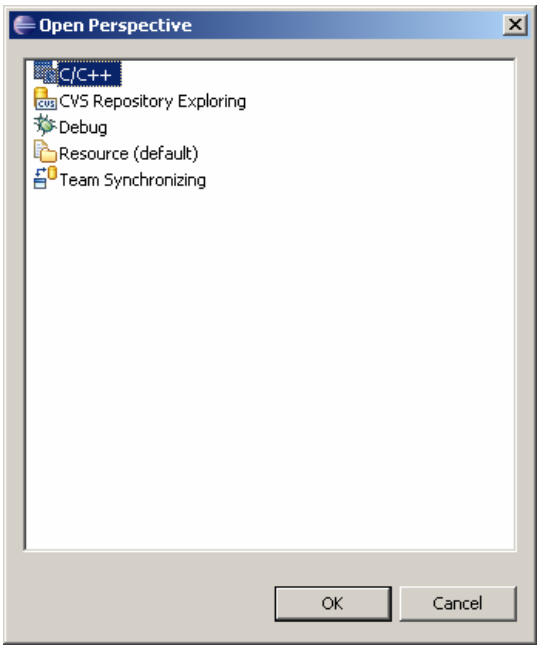
Now click at the "Workbench" icon on the right edge of the window, and the next window will look like:



This is the “Resource Perspective”, but we need to open the “C/C++ Perspective”. Use “Window / Open Perspective / Other ...”

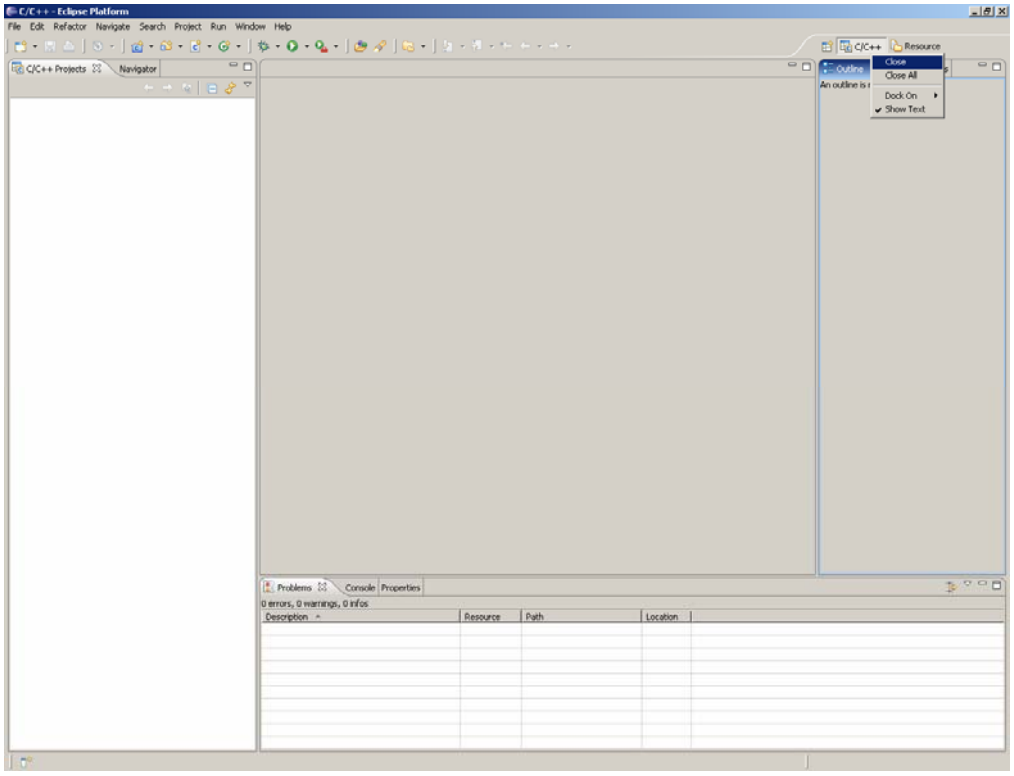


The next window will be opened:

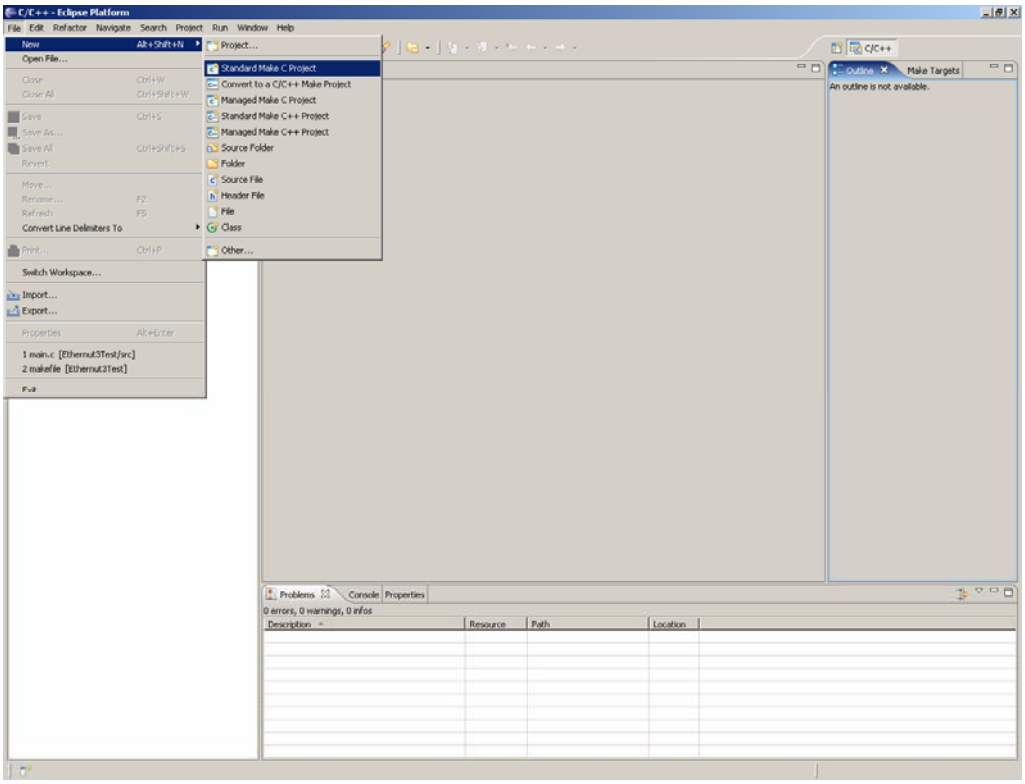


Select “C/C++” and press the “OK” button. Now the “C/C++” and “Resource Perspective” are part of the window:

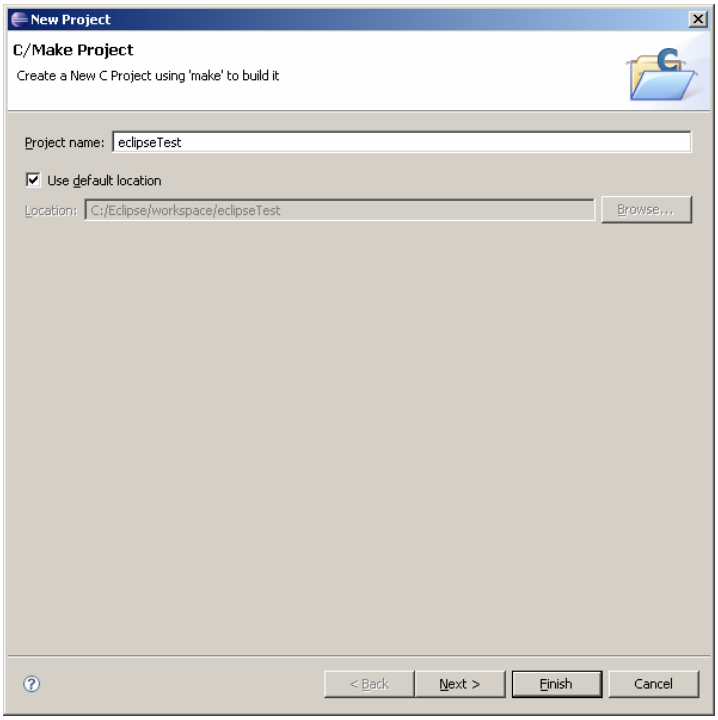
We only need the “C/C++ Perspective” and may close the Resource one. Therefore right click on the Resource perspective in the right edge of the window and closes the Perspective:



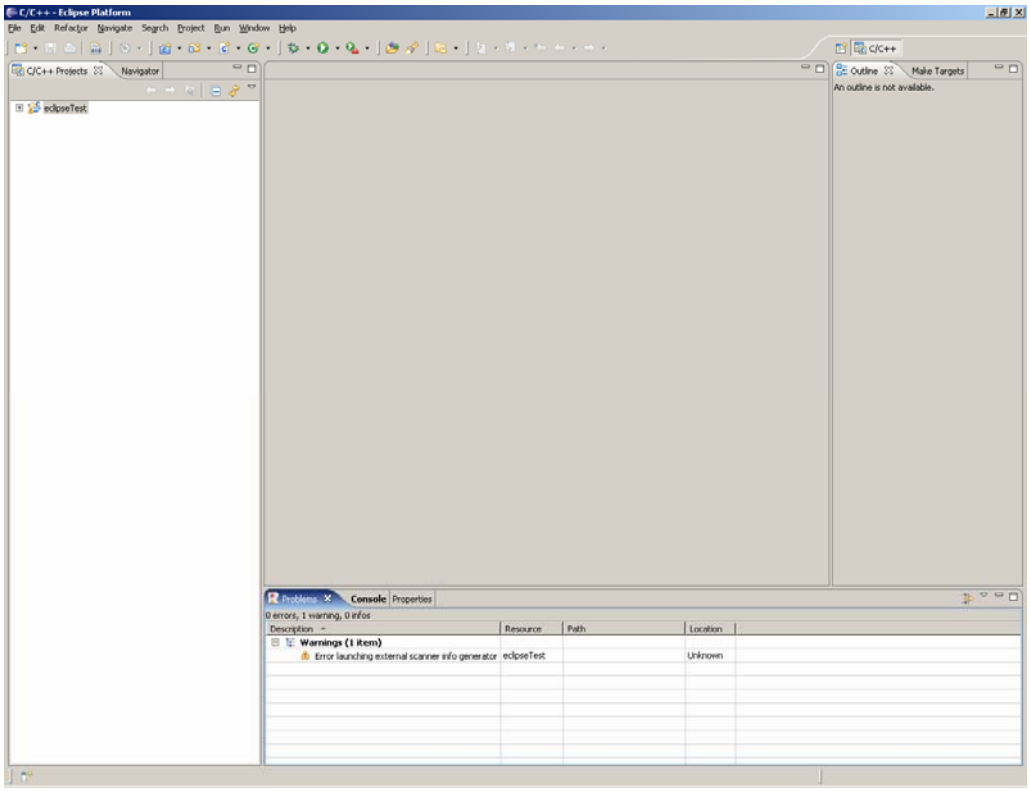
To create an Eclipse project use: “File/New/Standard Make C Project”



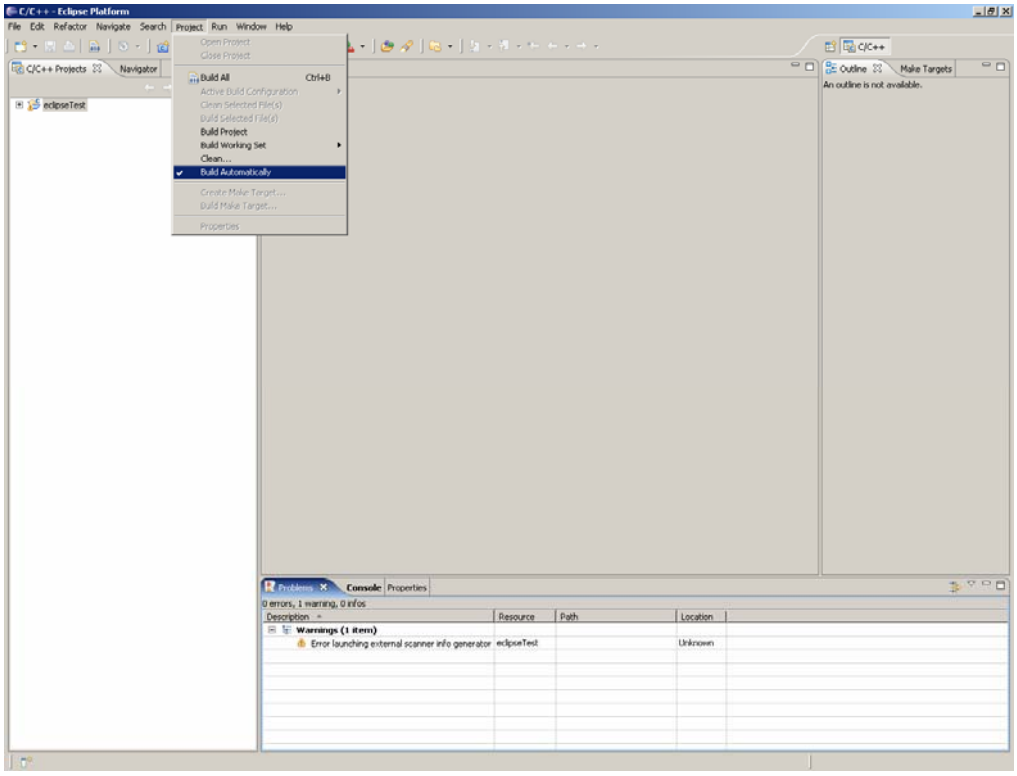
The next window will be show up:



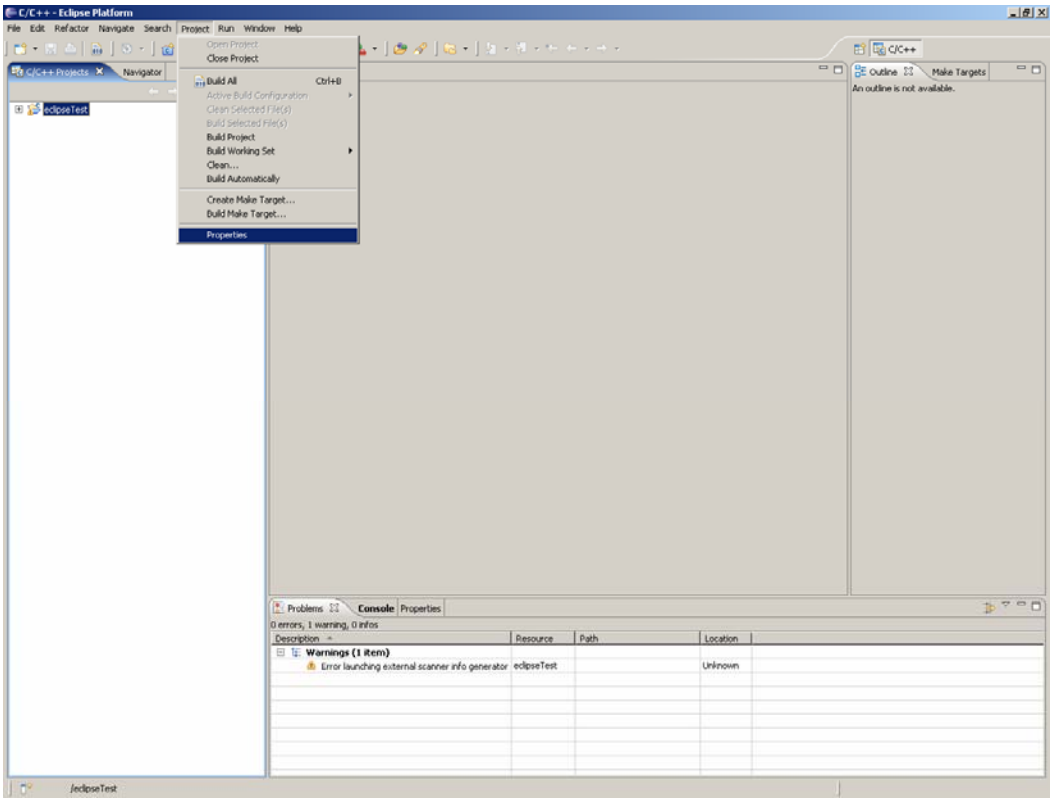
Here you will see that the "Location" is your workspace directory you created some steps above. Type in the "Project name" and press the "Finish" button. Now the Eclipse window should look like:



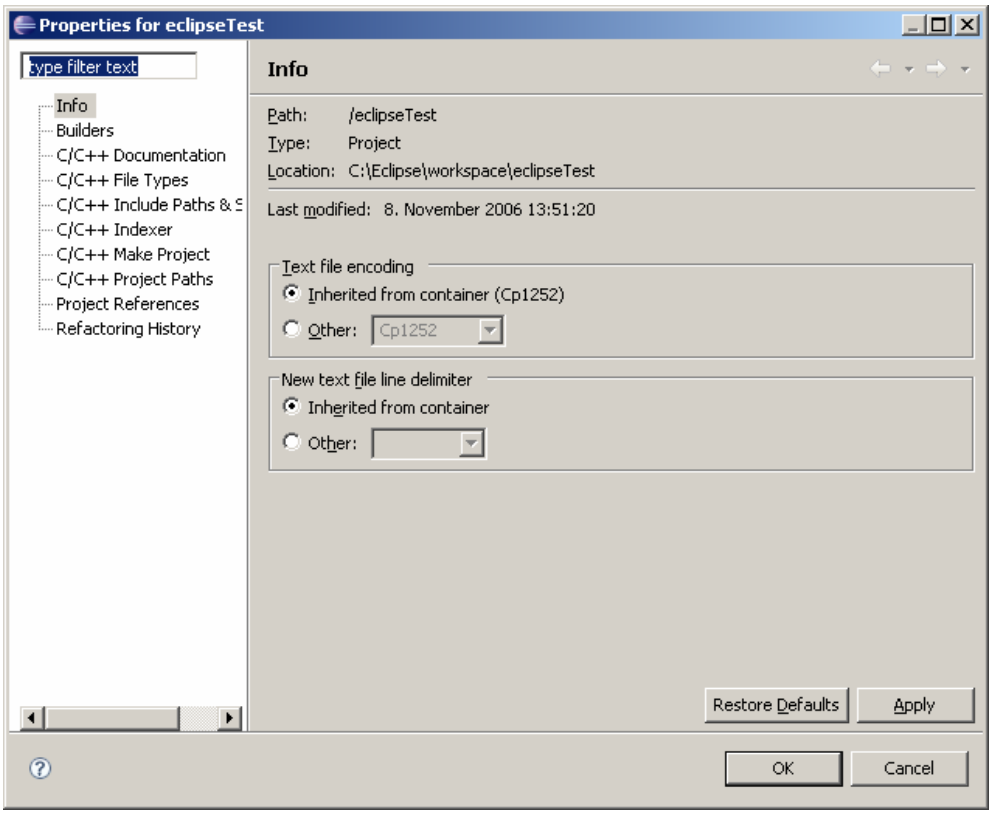
You just created your first Eclipse project. You now need to configure it. Use “Project” menu and remove the checkmark at “Build Automatically”.



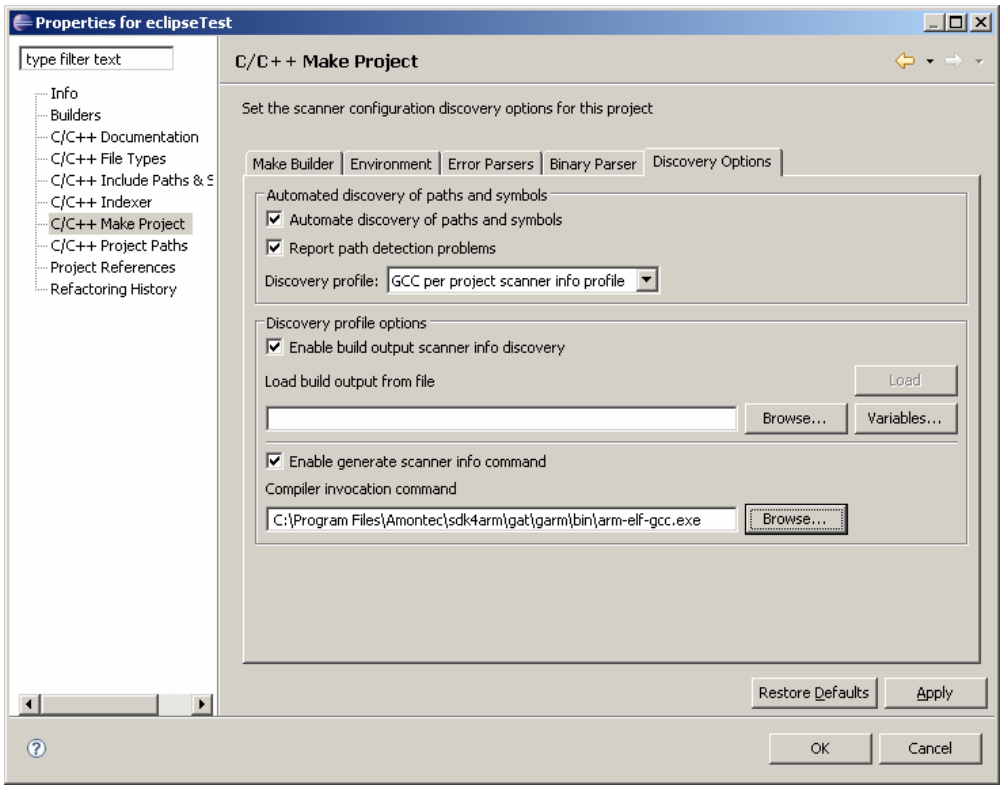
Use the “Project” menu again and select “Properties”:



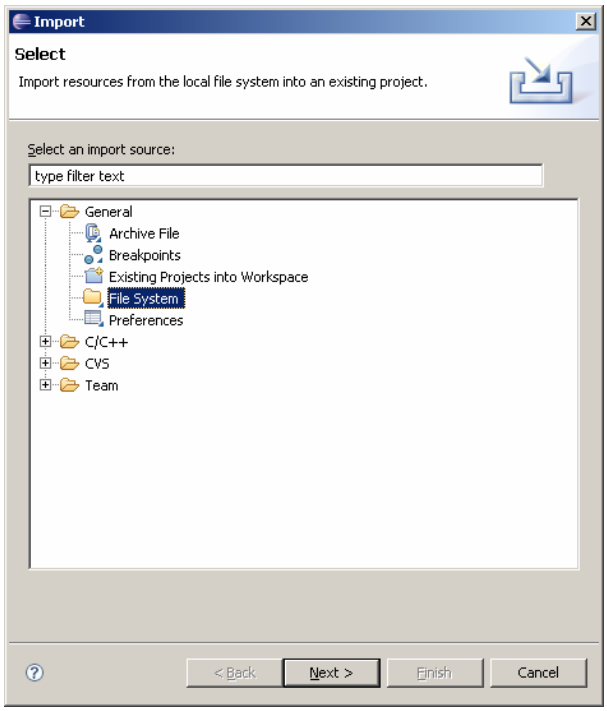
A new window will be opened:



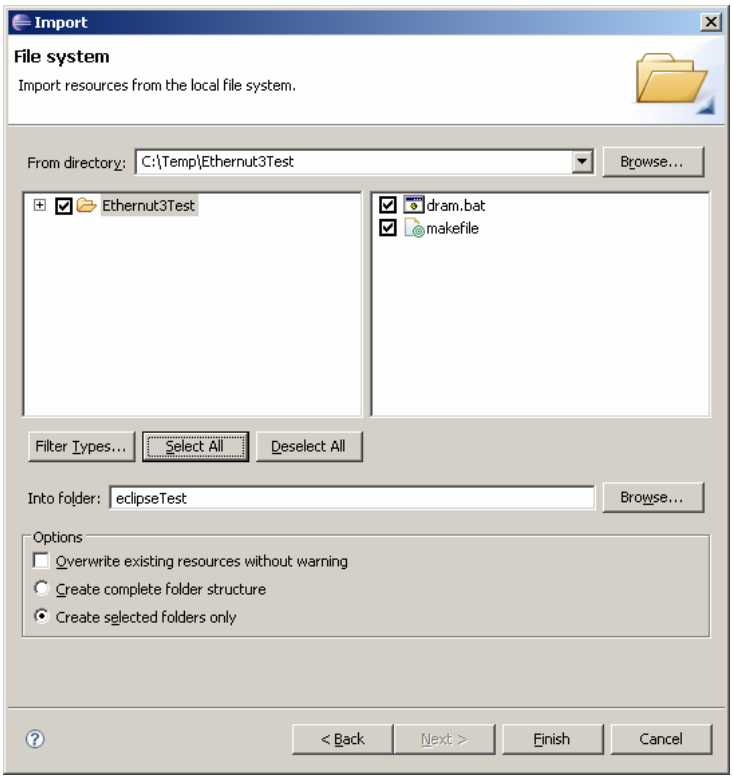
Select "C/C++ Make Project" and go to the "Discovery Option" tab:



Change here the “Compiler invocation command” to the Amontec sdk4arm compiler executable arm-elf-gcc.exe. Press the “OK” button to finish this dialog. At that time the project is empty and you must import the project files to Eclipse. Use the “File / import…” menu and select the “File System” as input source.

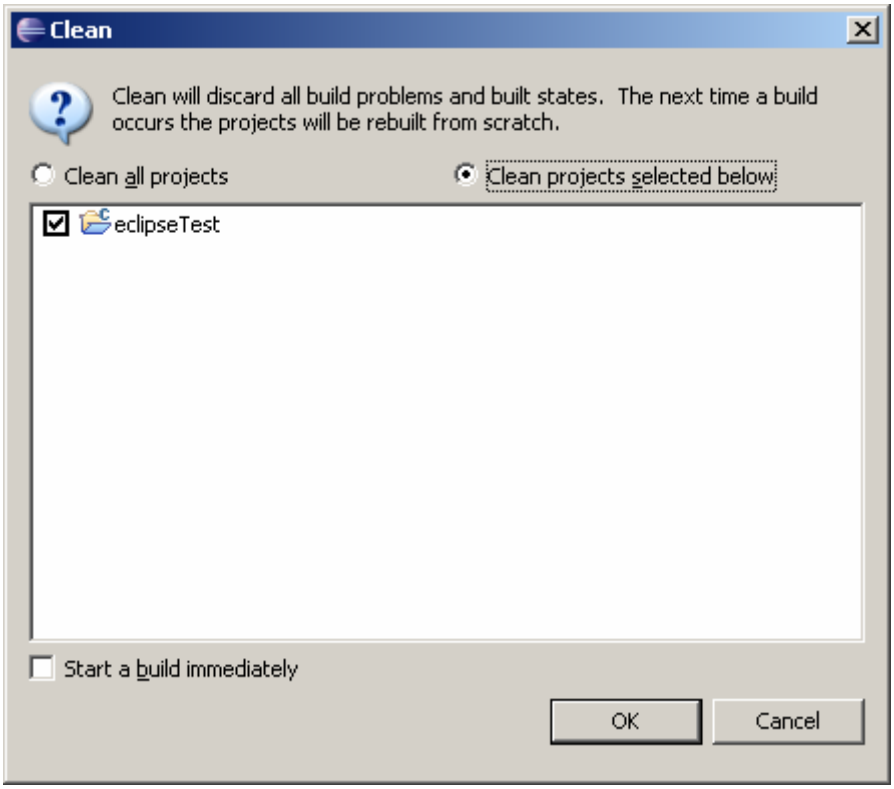


Press the “Next” button to open the “Import” dialog:

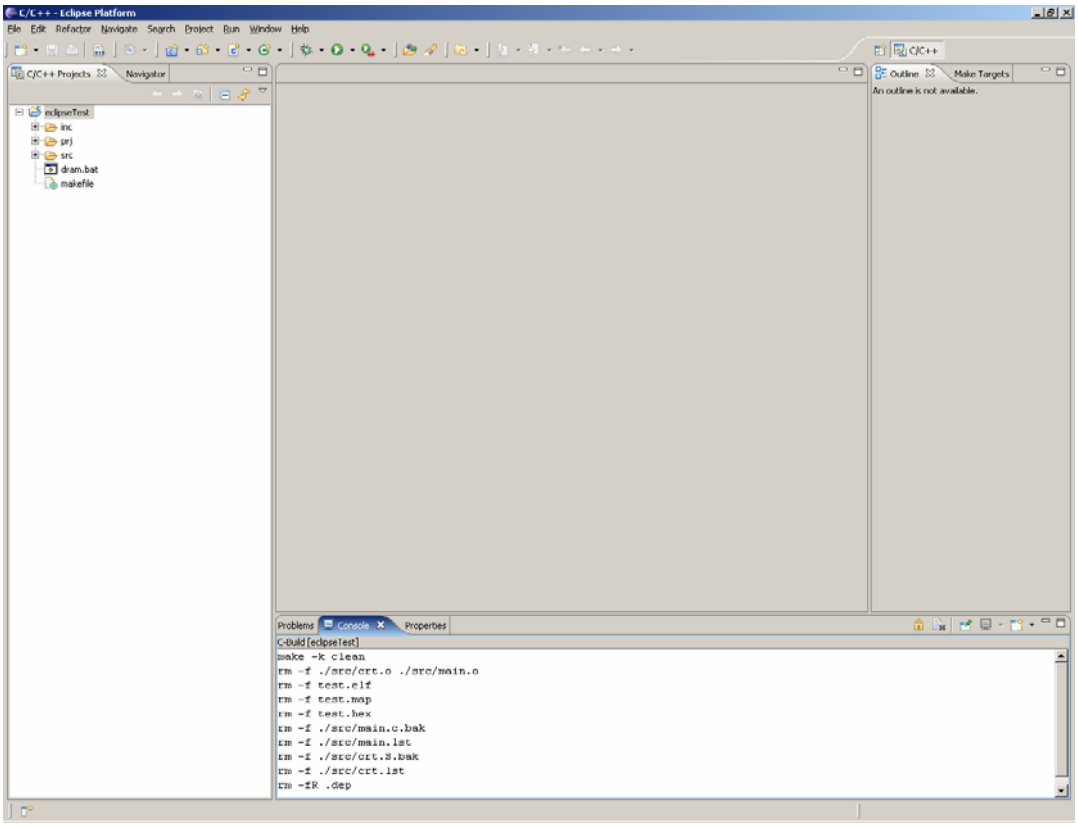


In order to test the environment development an example was downloaded. It can be found at this address: http://www.yagarto.de/download/yagarto/Ethernut3Test.zip and was downloaded to the following directory: "C:\Temp\Ethernut3Test". Select this directory for "From directory" and select all file with the checkmark. Press "finish" to close the dialog.

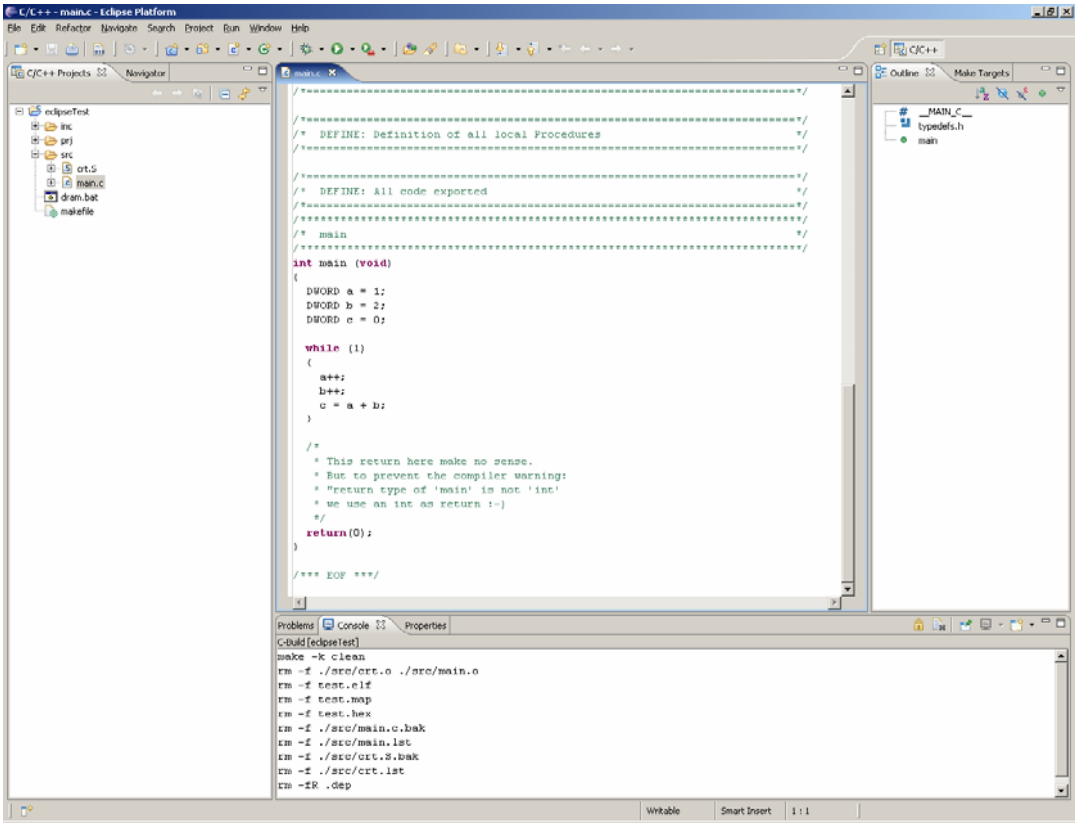
Now let's "clean" the project. For this purpose use "Project / Clean…". Set the options according to the next window:



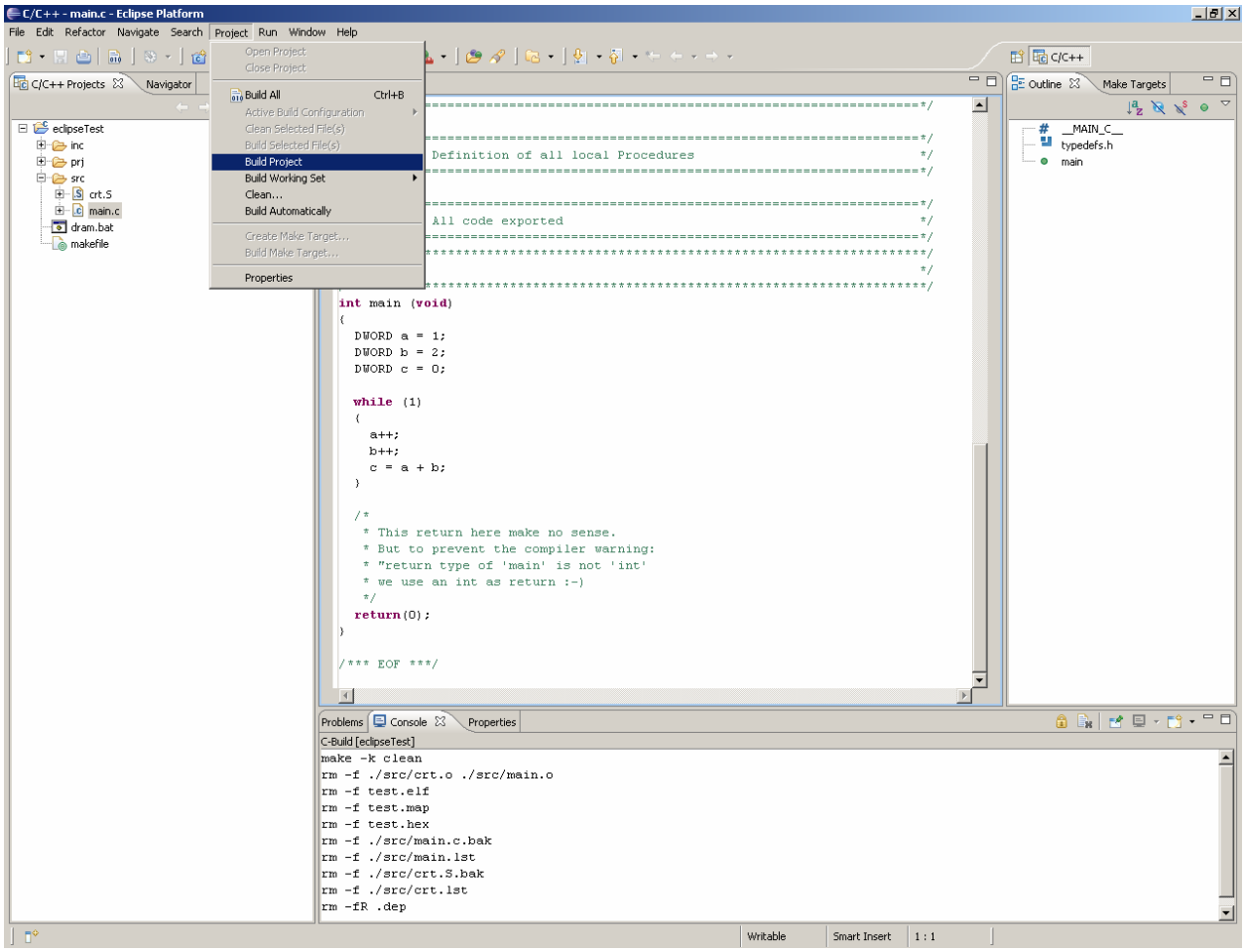
Remove the checkmark at "Start a build immediately", and press "OK". The "Console" will show the result of the "Clean" command:



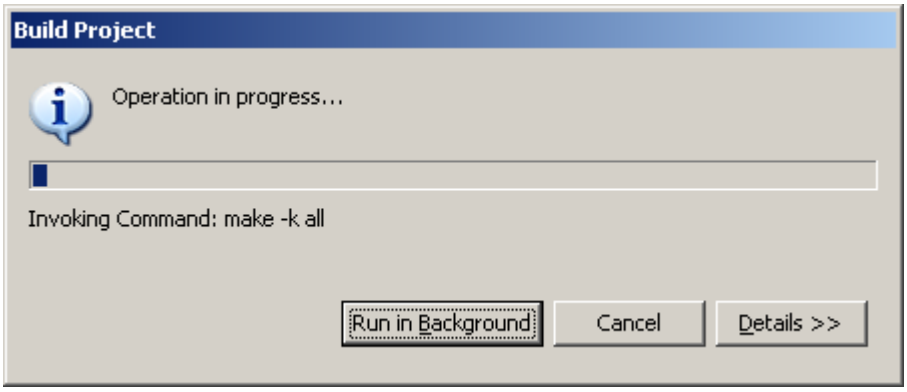
By the way, it is possible to open any source file by simply double clicking on it. The editor will open up:



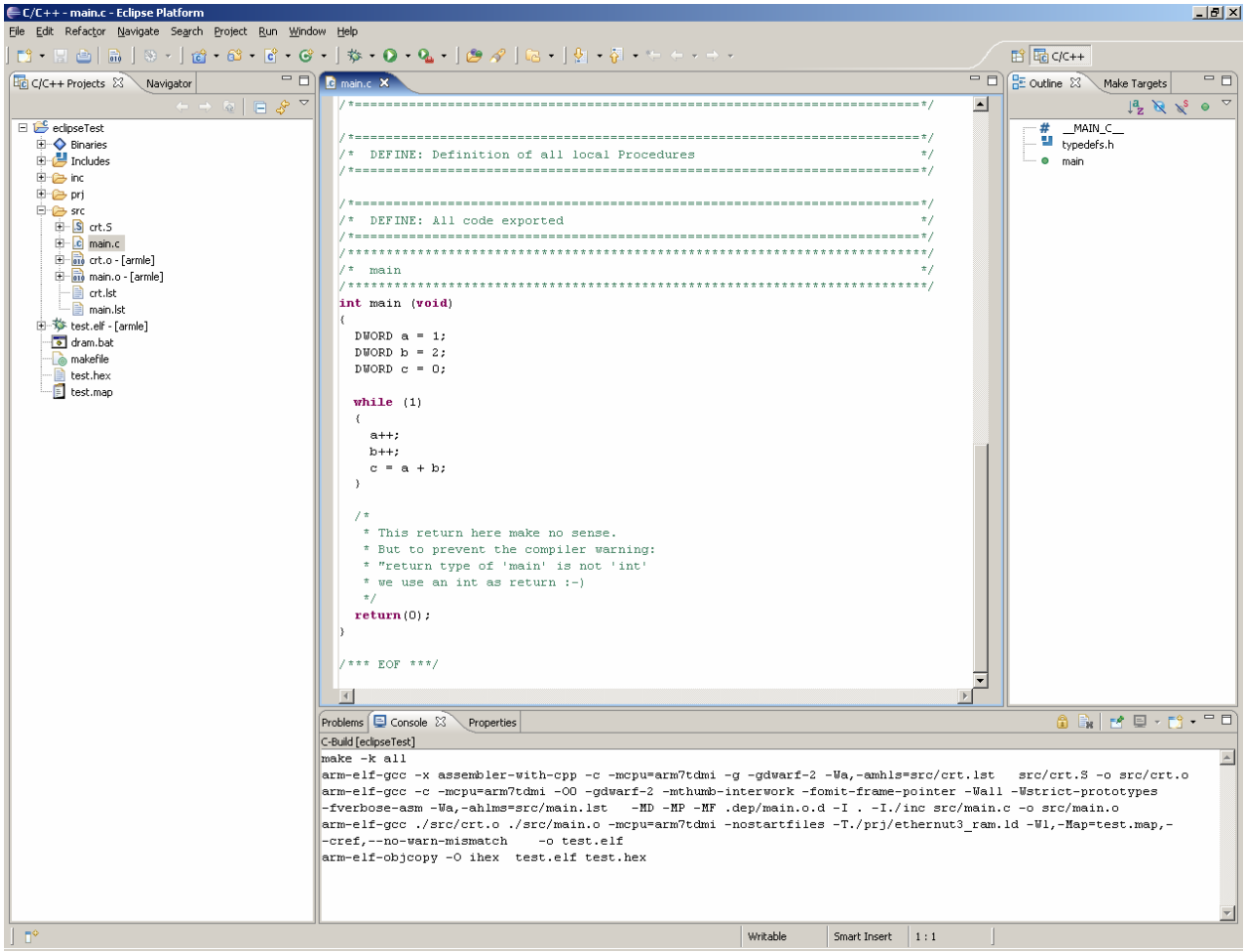
But do not change the file. Now let's build the project using "Project / Build Project":



The project will be build, and the window look like:

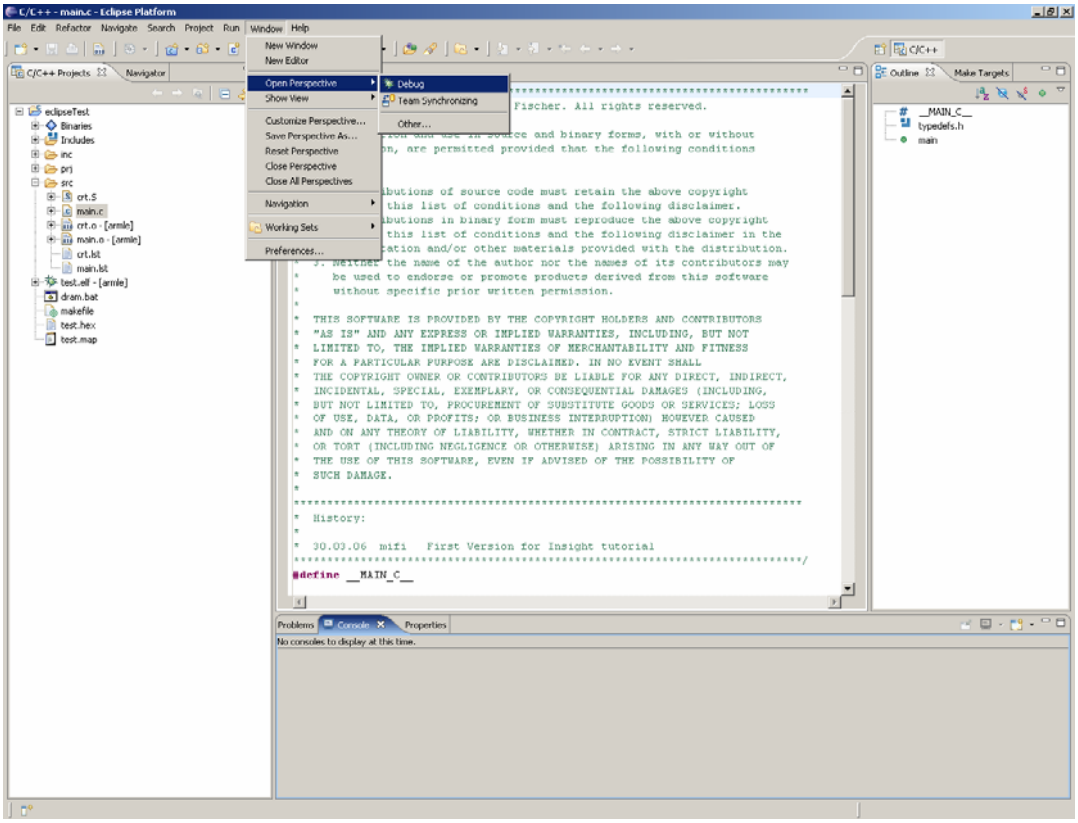


At the same time it is possible to see the result of the build process inside the "Console" window. After a successful build process, an "elf" file was created:

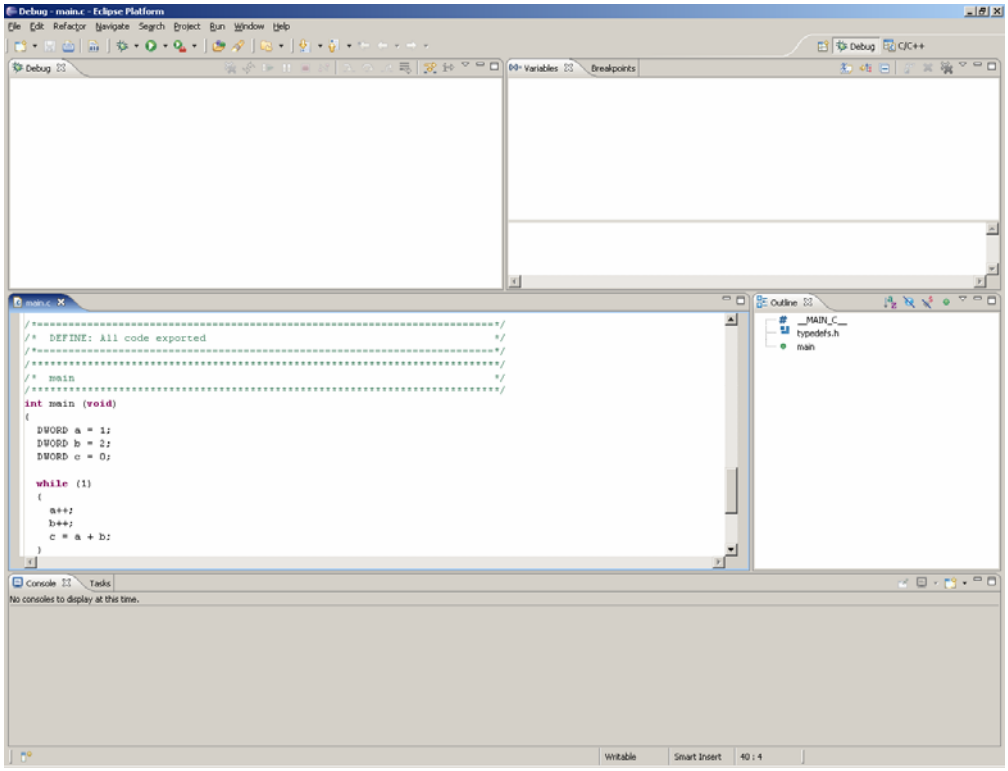

C/C++ - main.c - Eclipse Platform
File Edit Refactor Navigate Search Project Run Window Help
C/C++
C/C++ Projects
Navigator
eclipseTest
Binaries
Includes
inc
prj
src
crt.S
main.c
crt.o - [armle]
main.o - [armle]
crt.lst
main.lst
test.elf - [armle]
dram.bat
makefile
test.hex
test.map
main.c
/*===*/
/* DEFINE: Definition of all local Procedures   */
/*===*/
/*===*/
/* DEFINE: All code exported   */
/*===*/
/***/
/* main   */
/***/
int main (void)
{
DWORD a = 1;
DWORD b = 2;
DWORD c = 0;
while (1)
{
a++;
b++;
c = a + b;
}
/*
* This return here make no sense.
* But to prevent the compiler warning:
* "return type of 'main' is not 'int'
* we use an int as return :-)
*/
return(0);
}
/*** EOF ***/
Outline
Make Targets
__MAIN_C__
typedefs.h
main
Problems
Console
Properties
C-Build [eclipseTest]
make -k all
arm-elf-gcc -x assembler-with-cpp -c -mcpu=arm7tdmi -g -gdwarf-2 -Wa,-amhls=src/crt.lst src/crt.S -o src/crt.o
arm-elf-gcc -c -mcpu=arm7tdmi -O0 -gdwarf-2 -mthumb-interwork -fomit-frame-pointer -Wall -Wstrict-prototypes
-fverbose-asm -Wa,-ahlms=src/main.lst -MD -MP -MF .dep/main.o.d -I . -I./inc src/main.c -o src/main.o
arm-elf-gcc ./src/crt.o ./src/main.o -mcpu=arm7tdmi -nostartfiles -T./prj/ethernut3_ram.ld -Wl,-Map=test.map,-
-cref,--no-warn-mismatch -o test.elf
arm-elf-objcopy -O ihex test.elf test.hex
Writable
Smart Insert
1 : 1

A.2 Debugger configuration

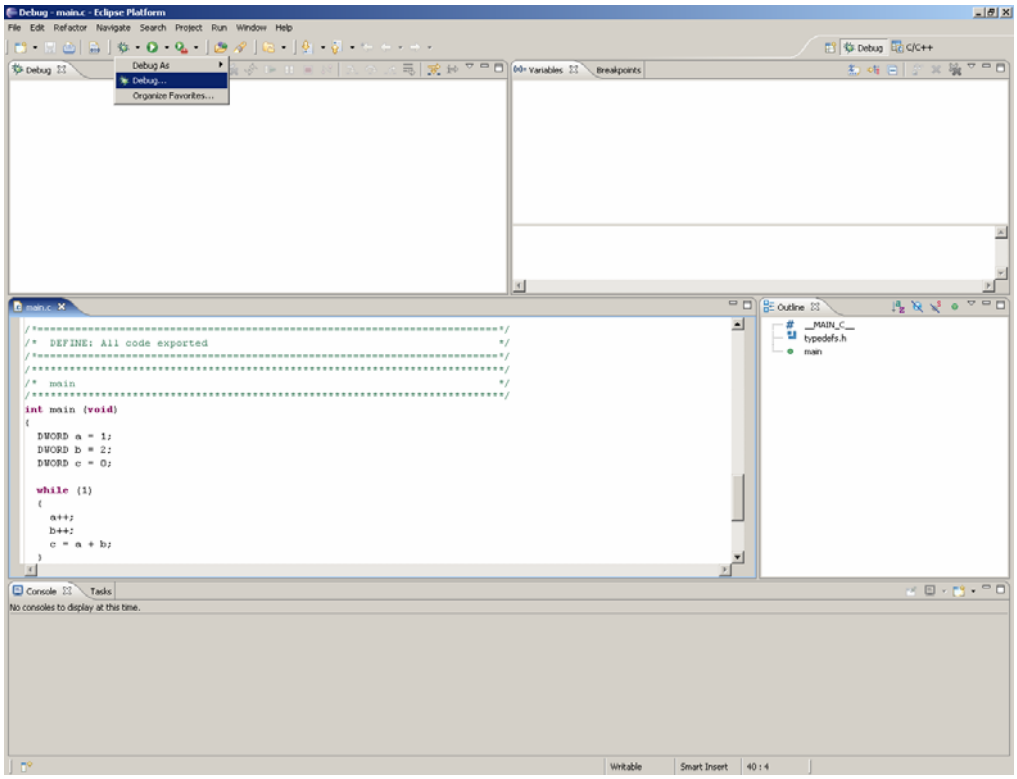
For the debug process it is required to open the "Debug perspective". For this purpose use "Window / Open Perspective / Debug":



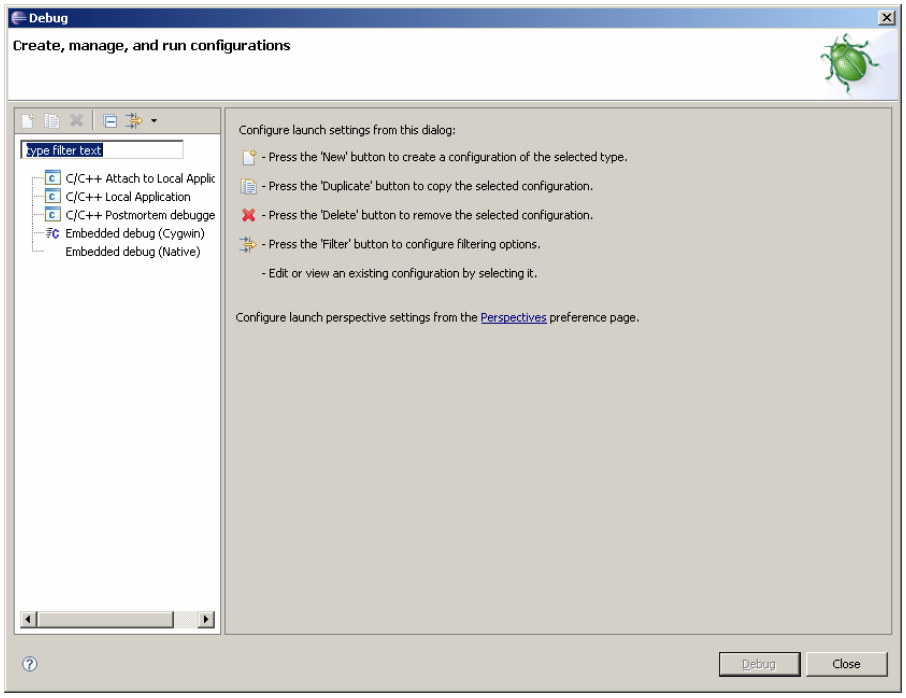
The new perspective will be opened. It looks like:



It is time to configure the debugger. The configuration can be reached by clicking on the down arrow at the "insect" button. That brings up a pull-down menu and select "Debug…":



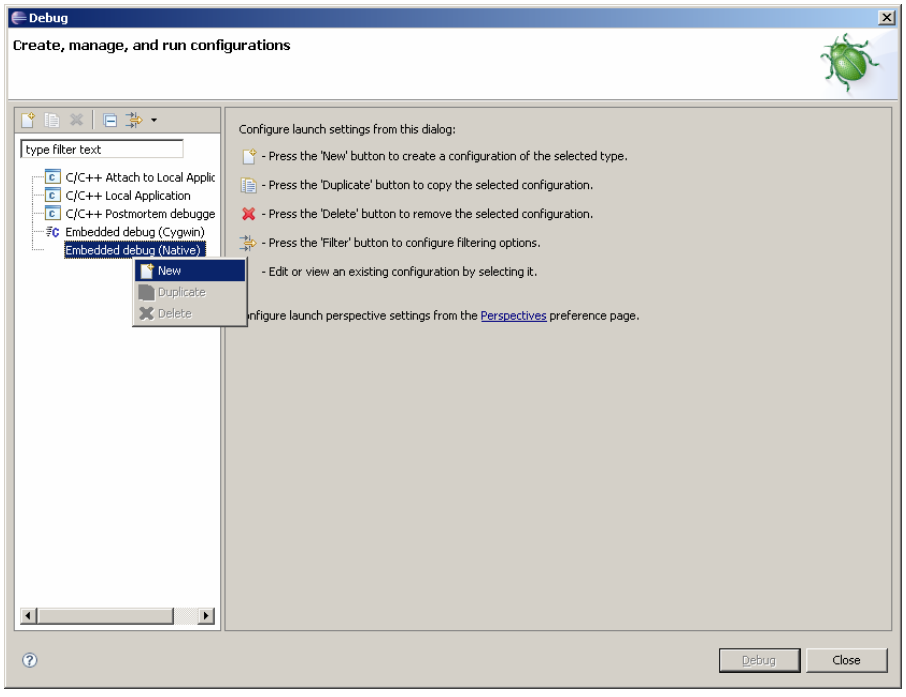
The next window will look like:



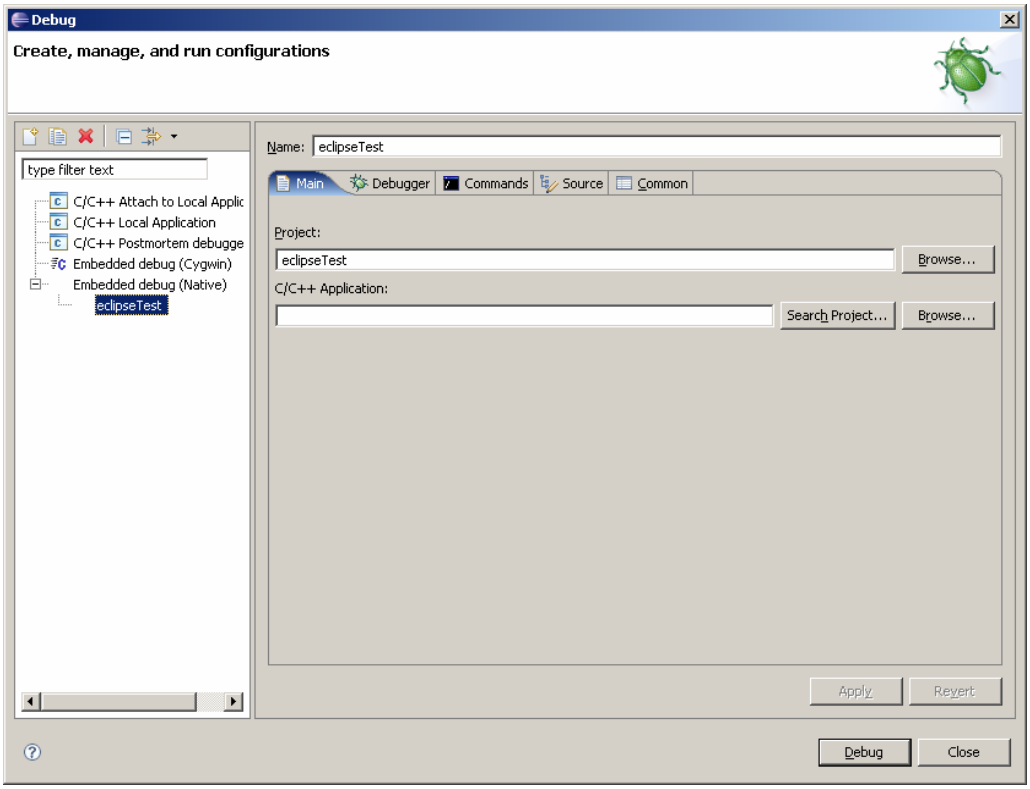
Now, it must be decided what kind of GDB is installer:

- Cygwin: the GDB need Cygwin to work
- Native: it means the GDB was compiled for a Windows host.

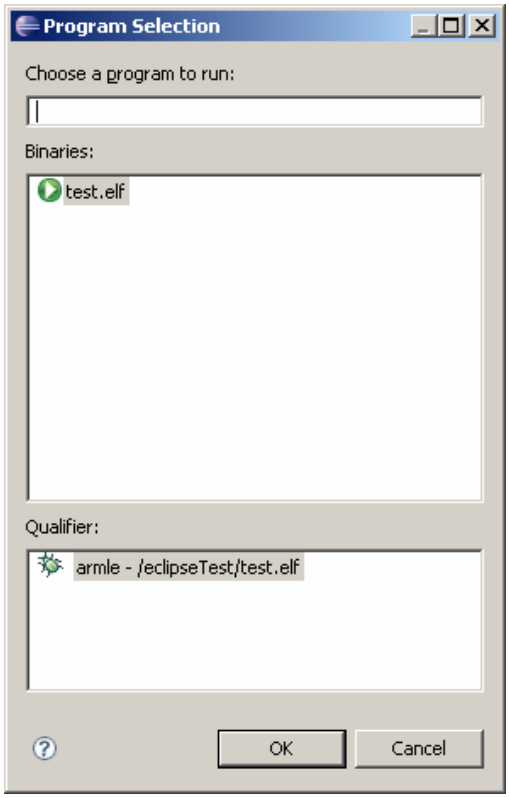
The GDB provided by the SDK4ARM from Amontec is based on windows. Right click on the "Native" section and press the "New" button to create a configuration of selected type.



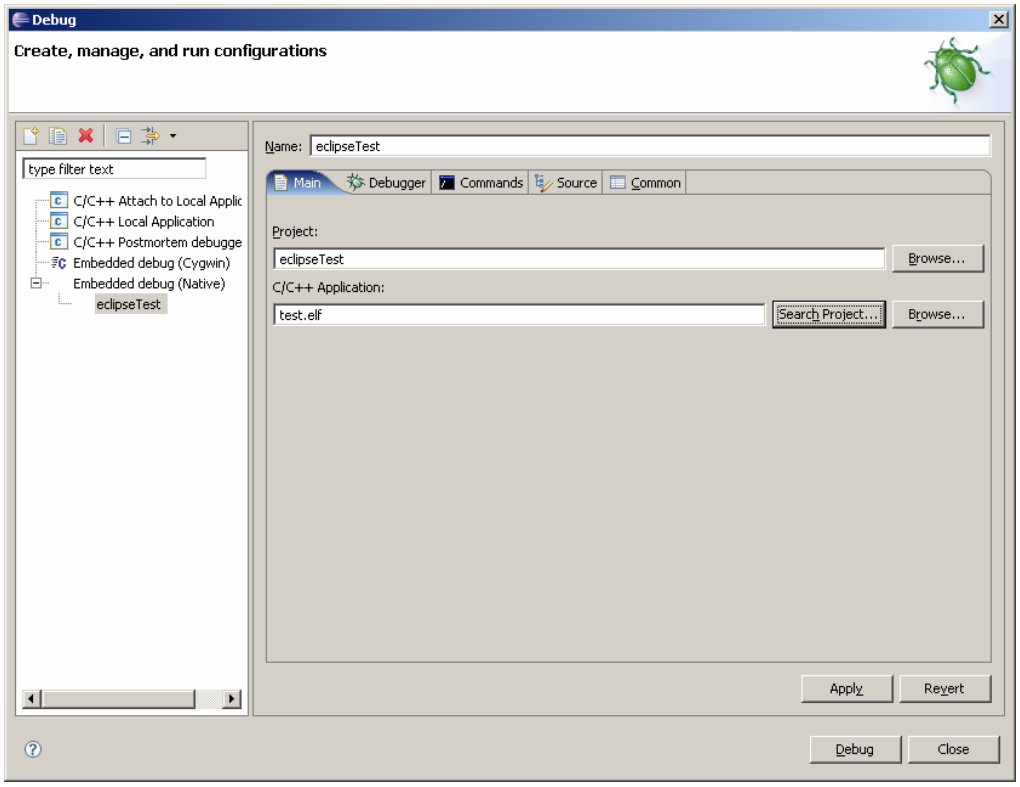
The configuration window will be changed and look like:



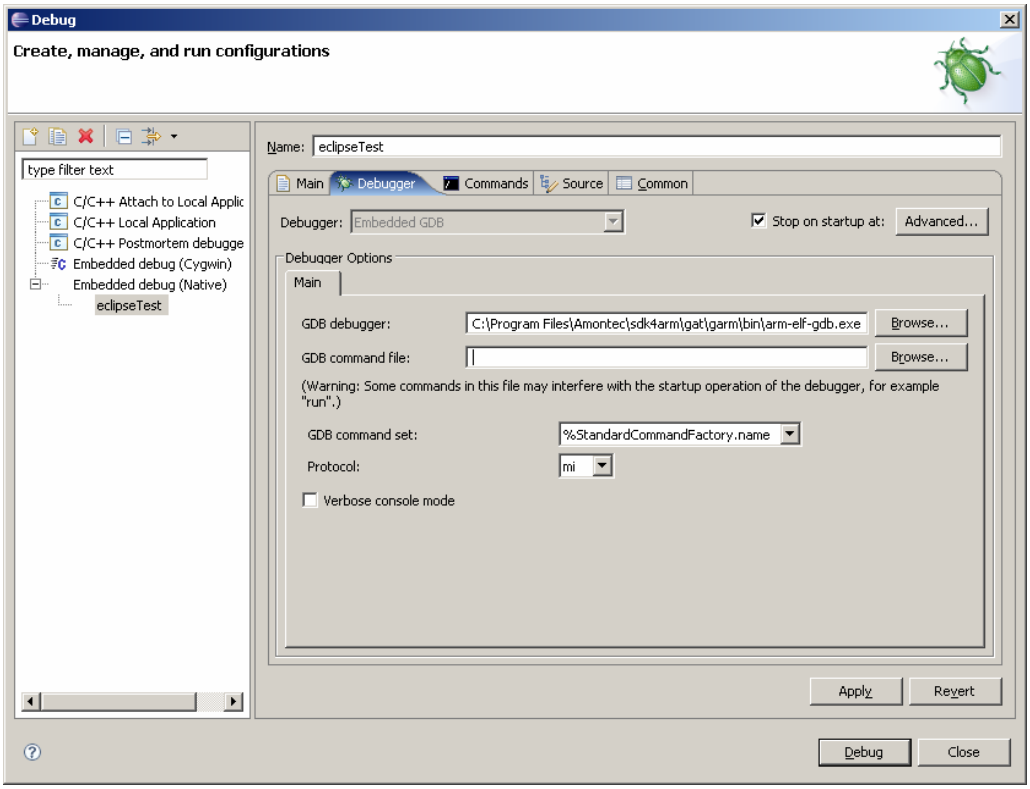
Click at the "Search Project…" button and select the "test.elf" file:



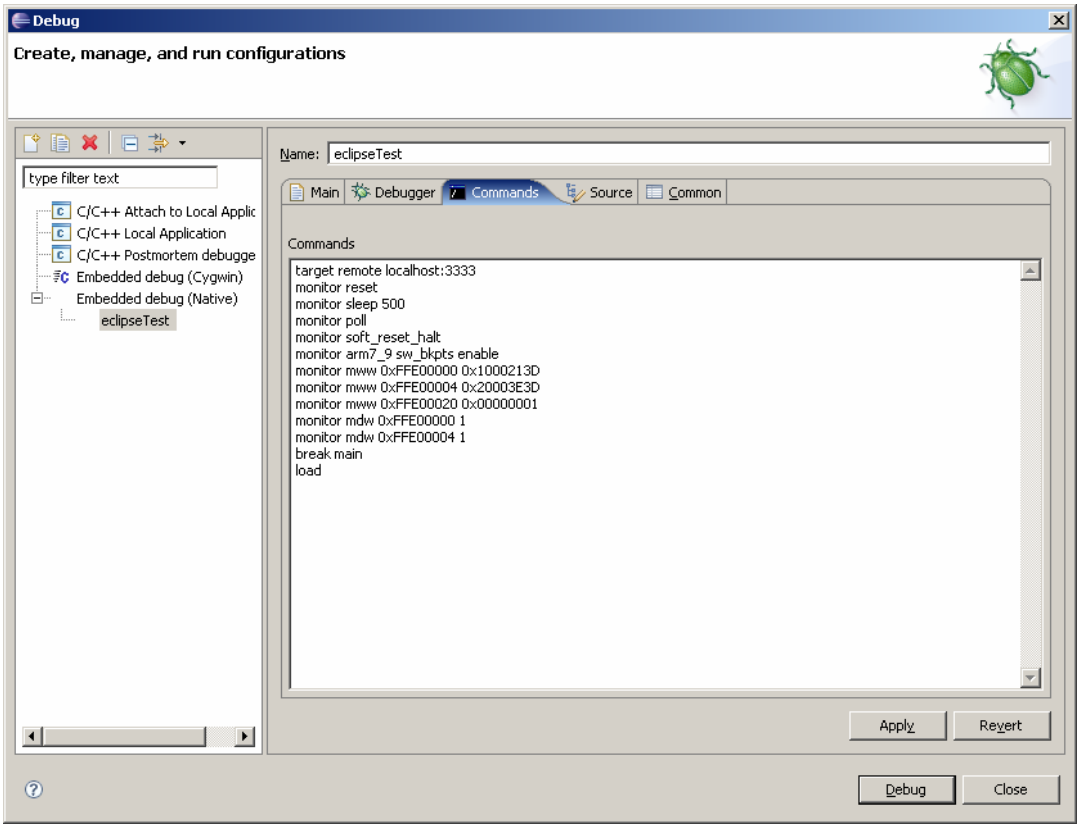
Press “OK” and the configuration window will look like:



Select the “Debugger” tab and use the “Browse…” button to set the “GDB debugger”:



Now change to the “Commands” tab and insert the debugger commands:

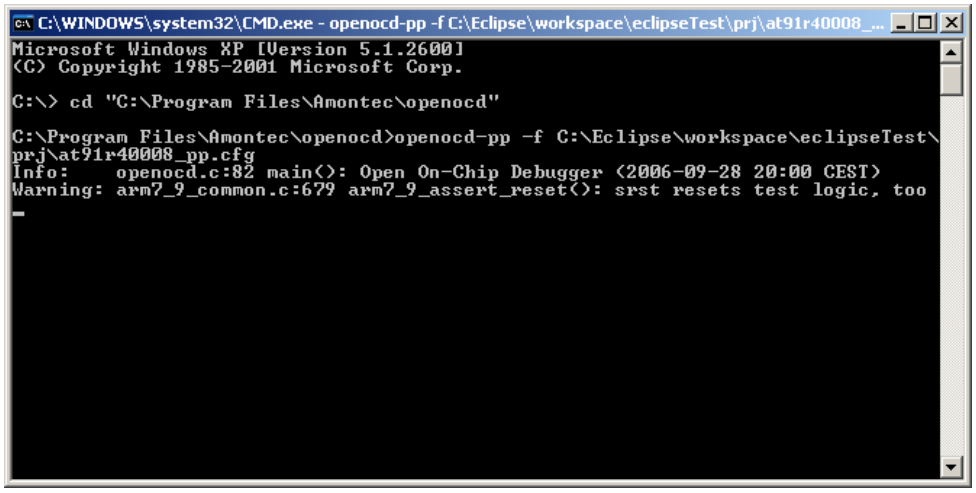


A file called “eclise_ram.gdb” is part of the example (.\prj\eclipse_ram.gdb). Copy and paste the contents of the eclipse_ram.gdb text into the “Initialize Commands” window. Now the configuration is done and the dialog should be closed with “Apply” and “Close” button.

A.3 How to debug?

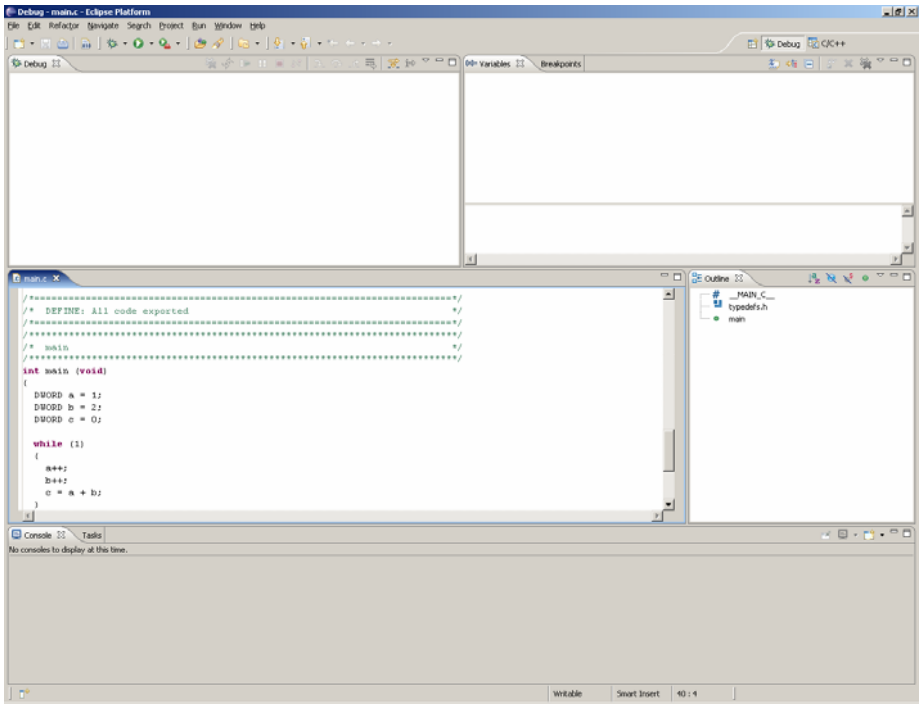
Open a command prompt and change to the OpenOCD directory. We need that command prompt for the OpenOCD debugger. In order to launch the debugger with the appropriate configuration file and with the parallel port; type:

"openocd-pp -f C:\Eclipse\workspace\eclipseTest\prj\at91r40008_pp.cfg"

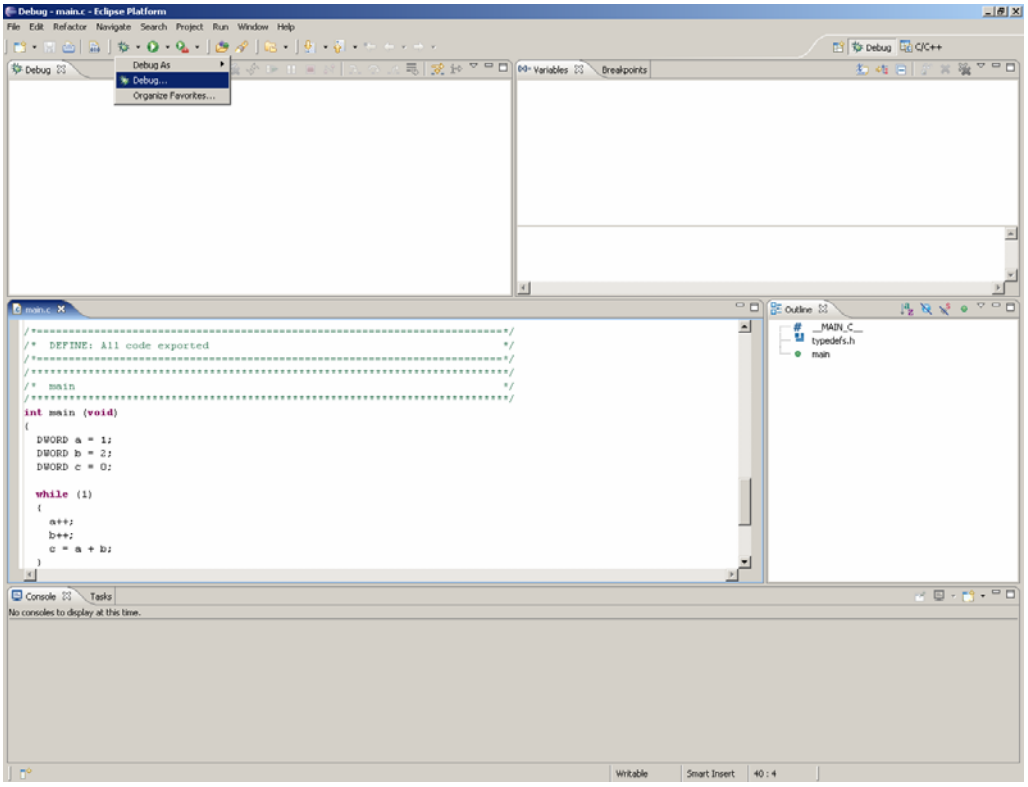


The warning is OK here. The OpenOCD debugger is started using the Chameleon pod, and configured for the AT91R40008 target, but also compatible with the AT91M55800A.

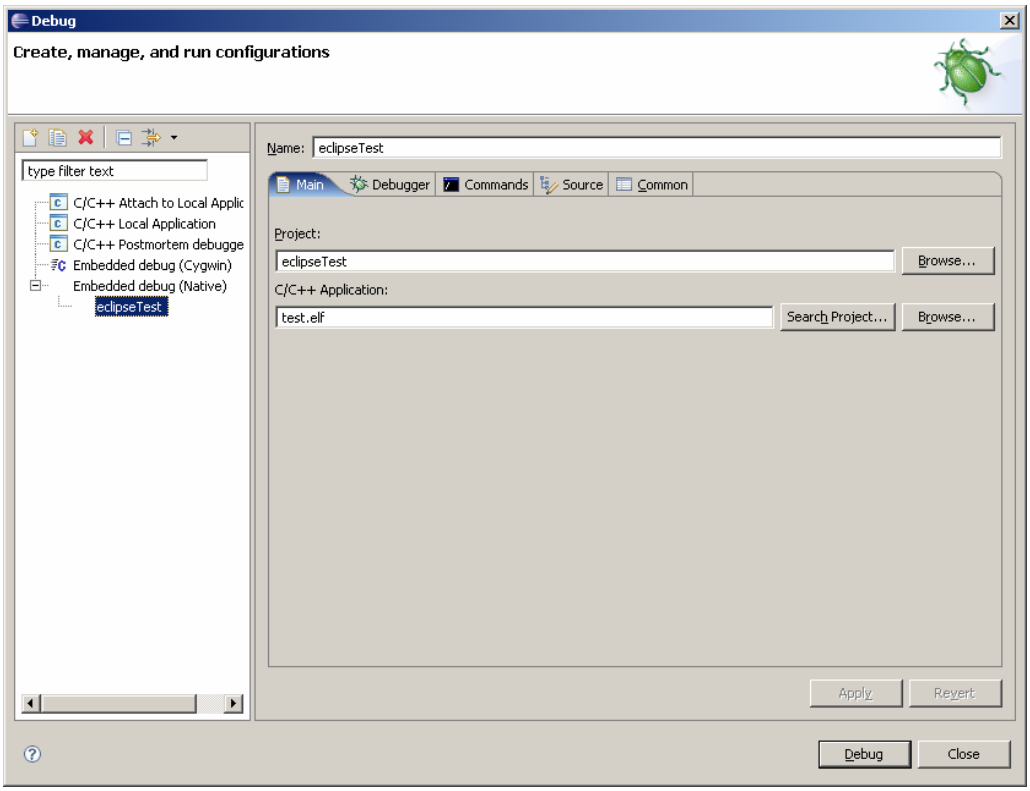
This command prompt is now blocked with the OpenOCD debugger, and we can go back to Eclipse. Eclipse should look like:



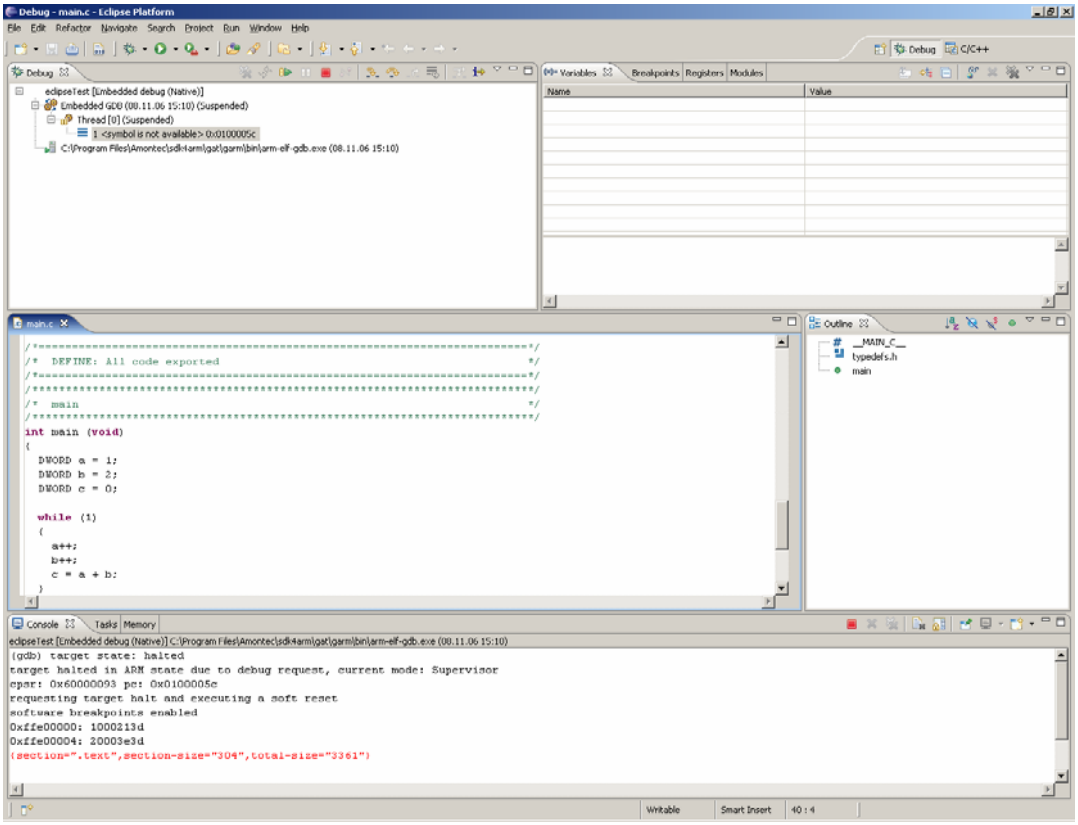
The debugger will be started the first time using the configuration menu:



Here the configuration prepared before should be found.

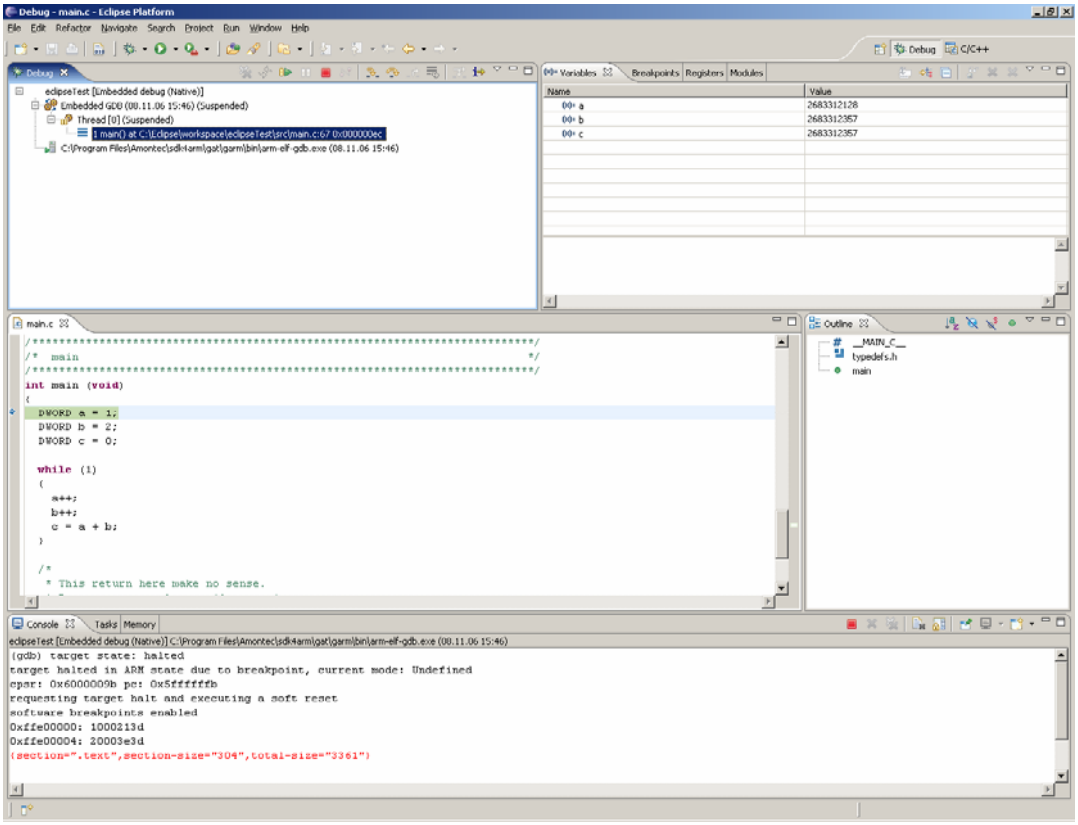


Press the “Debug” button to start the debugger. Eclipse should look like:



The debugger is running. The window at the bottom is the “GDB Debugger” command window. The upper left window is the “Debuf” window. Here some icons for stepping through the code are presented. In the debugger command the first beakpoint was set to NutMain. The debugger should no go to this breakpoint. Therefore press the “Resume” button:

Eclipse should now look like:

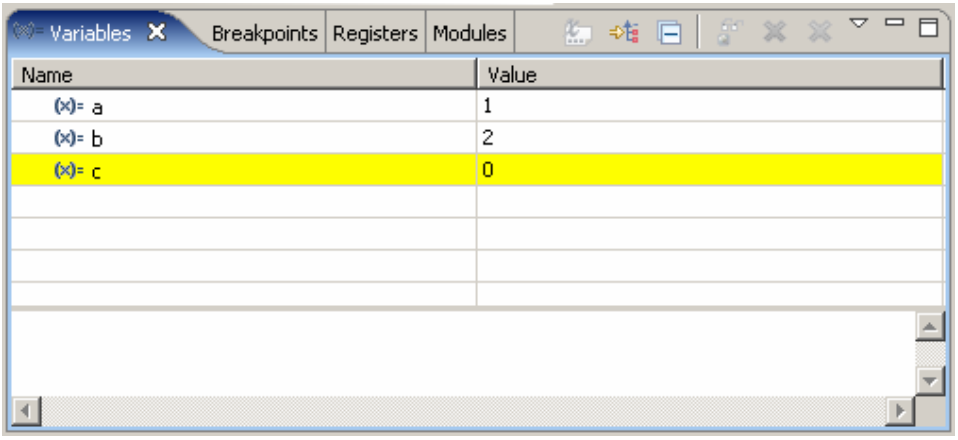


The highlighted line in the source window (main.c) is the actual line which will be executed in the next step. The upper right window shows the "Variable". It is possible to change the view to "Breakpoints", "Register" and "Modules". At this time the variable "a, b, c" are not yet initialized. The strange value of these variables does not matter.

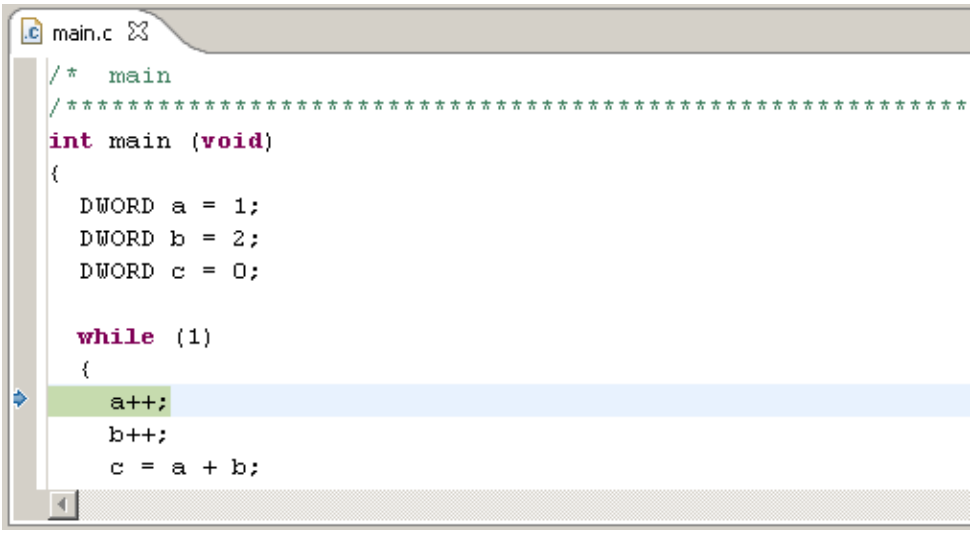
In the Debug window it is possible to step through the code with the following button:

	
Step into	Step over

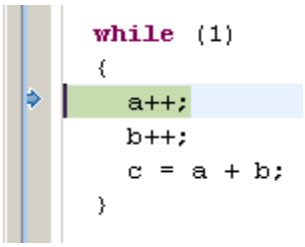
After three steps the “Variable” windows should look like:



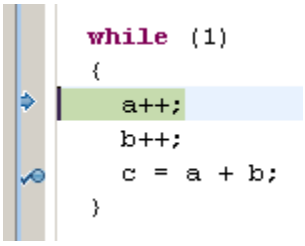
And the source window:



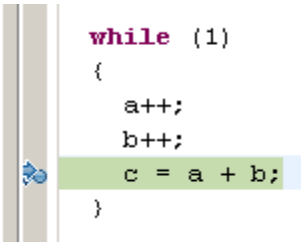
With the three steps from above, “a = 1, b = 2 and c = 0” was executed. The next line will be “a++”. The line which is highlighted is not yet executed. This will be done with the next step. It is possible to set a breakpoint by double clicking in the left grey area of the source window. The area where the blue arrow is:



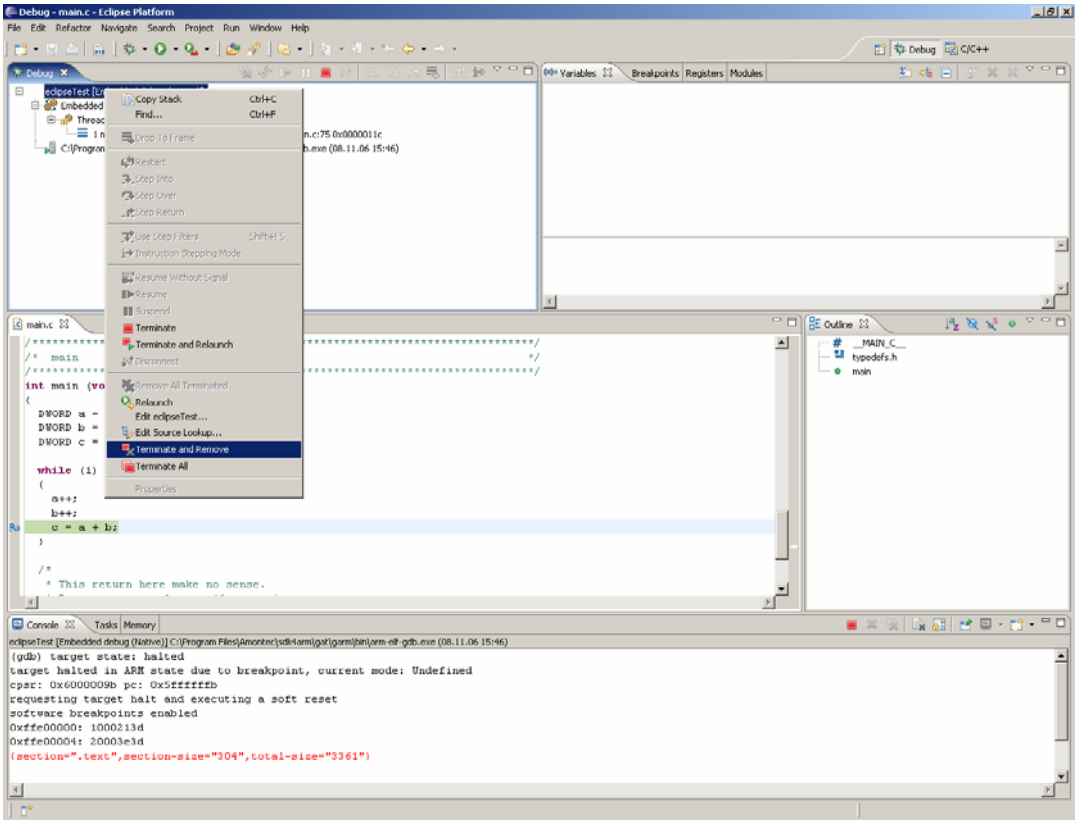
By double clicking in line of “c = a + b” a breakpoint will be set:



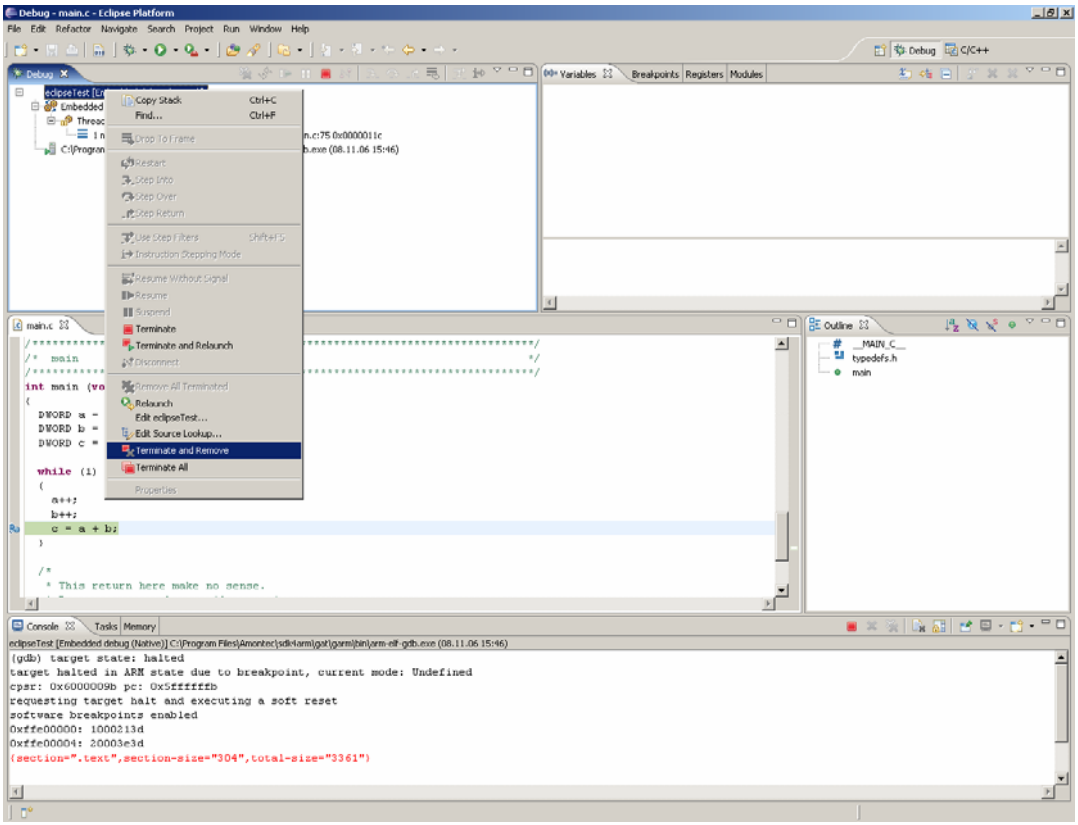
The blue checkmark with the blue dot is the new breakpoint. By double clicking on it again, the breakpoint will be removed. If the resume button will be pressed, the program will be executed and it will stop at the breakpoint.



The value of the variable was changed to “a = 2 and b=3”. It is possible to “Terminate and Remove” the debug task with a right click on “eclipseTest” inside the “Debug” window:



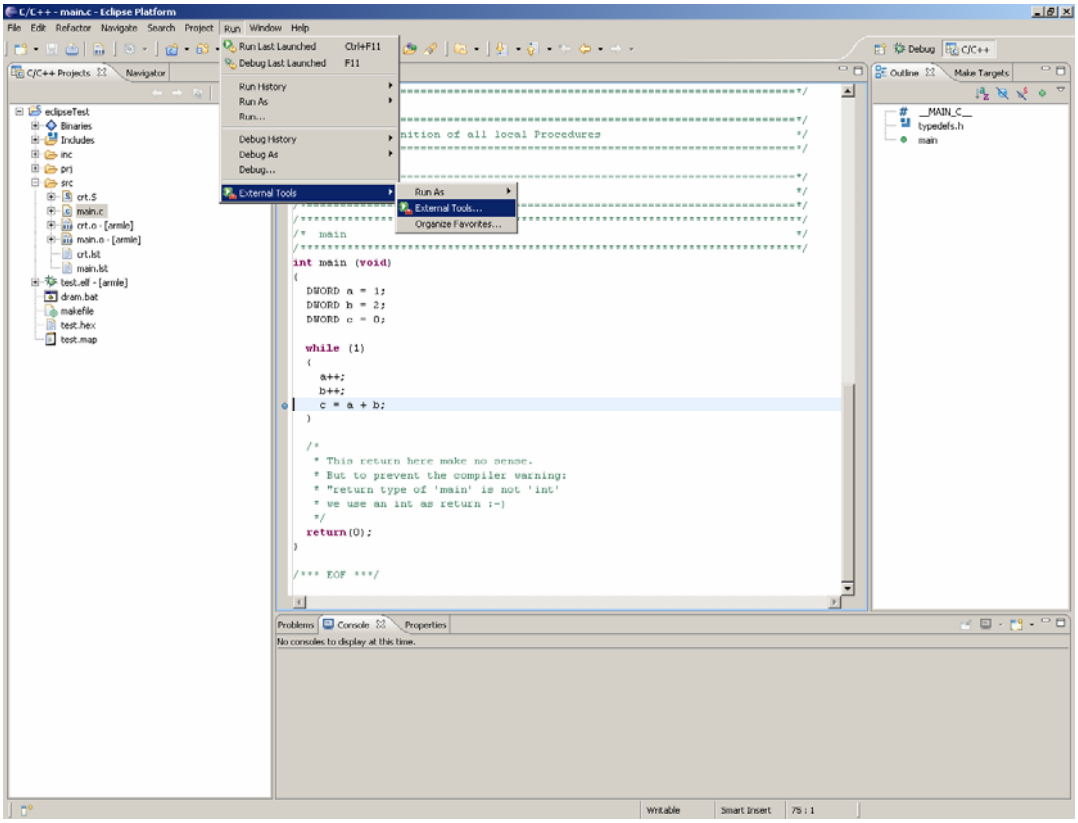
It is possible to start the debugger again by the “insect”, but at this time an entry for “eclipseTest” is available:



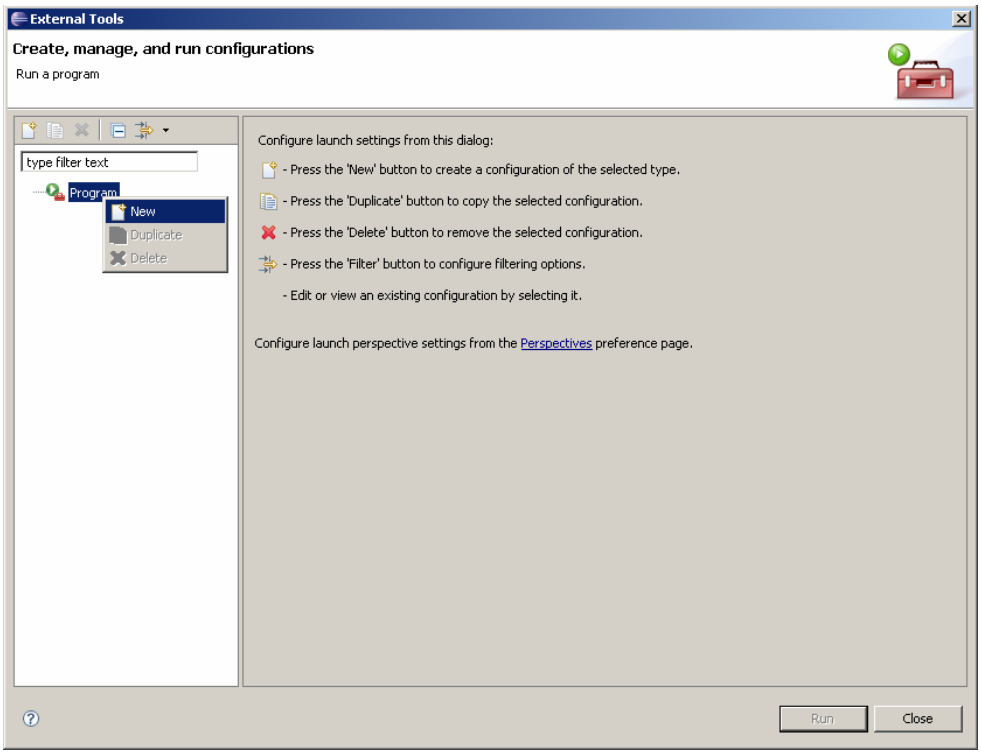
A.4 Install OpenOCD as an Eclipse External Tool

This section describes how to install OpenOCD as an Eclipse "External Tool" so that will be very convenient to start it from within the Eclipse environment.

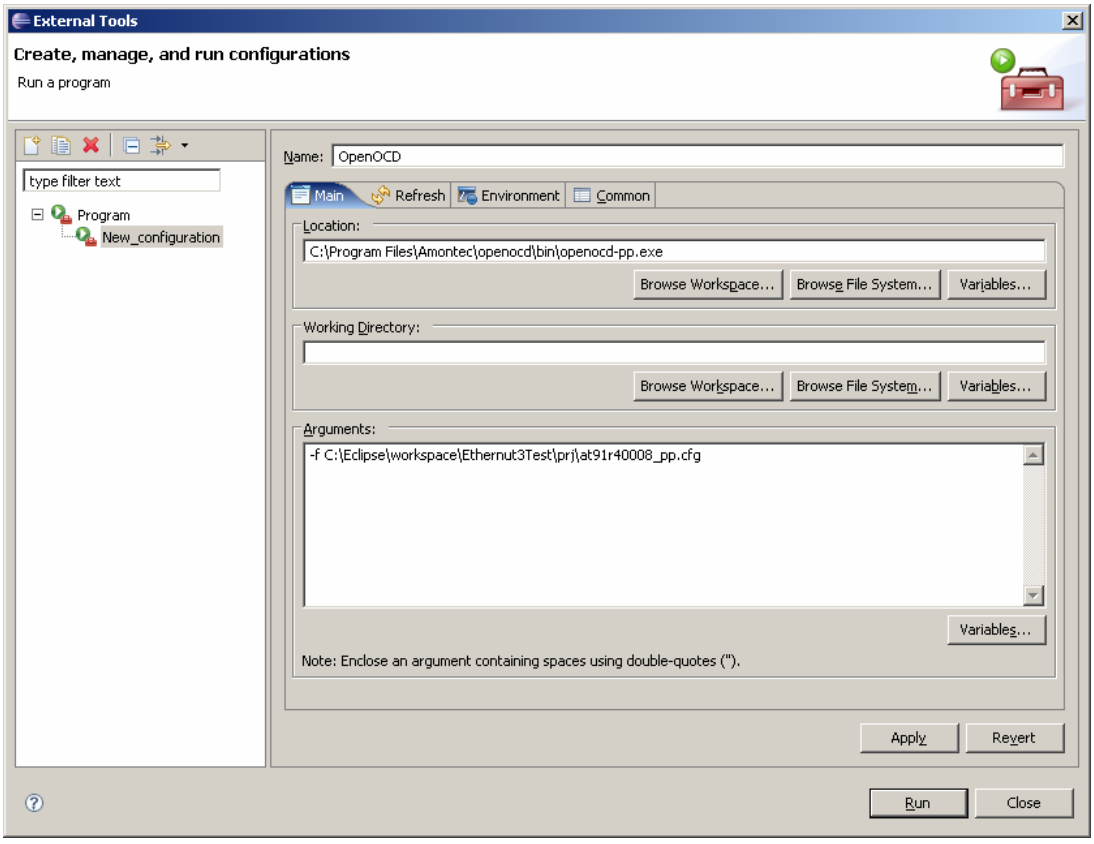
Click on "Run / External Tools / External Tools…":



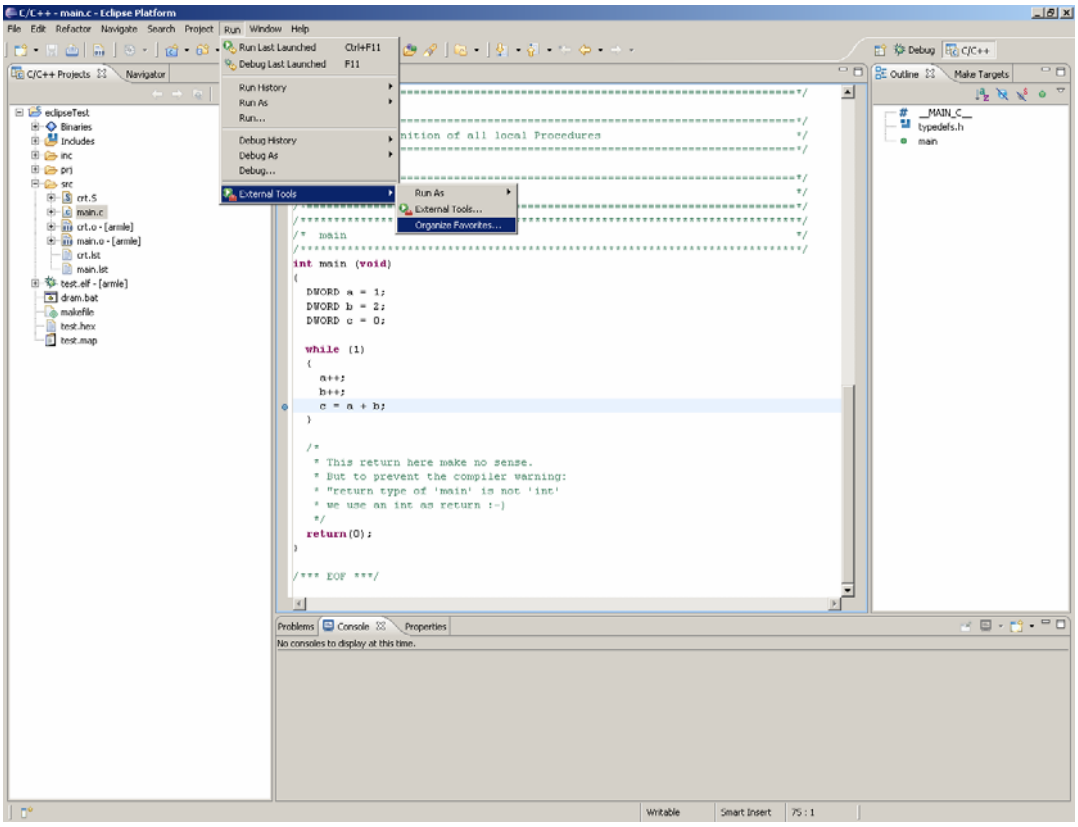
The "External Tools" window will appear. Right click on "Program" and then "New" to establish a new external tool.



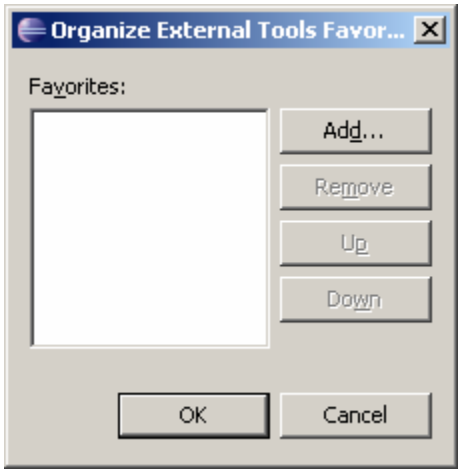
Fill out the "External Tools" from exactly as shown below. Click on "Apply" and "Close" to register OpenOCD as an external tool. The configuration file for the "wiggler" has been included in the arguments pane as reference. As said before, the configuration file, while designed for an AT91R40008 processor, will work fine for the AT91M55800A also.



There is one more step in defining OpenOCD as an external tool. This tool needs to be installed into the list of “favourite”. Click on “Run / External Tools / Organize Favorites…”

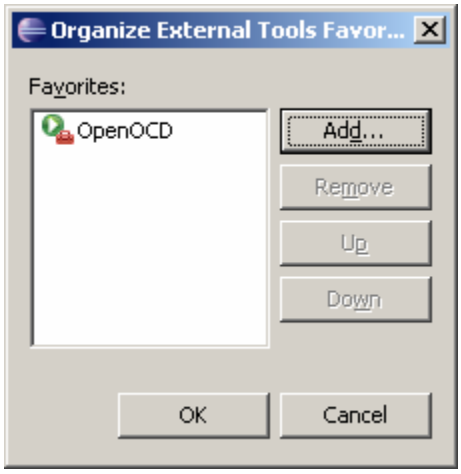


In the “Organize External Tools…” window, click on “Add…”.

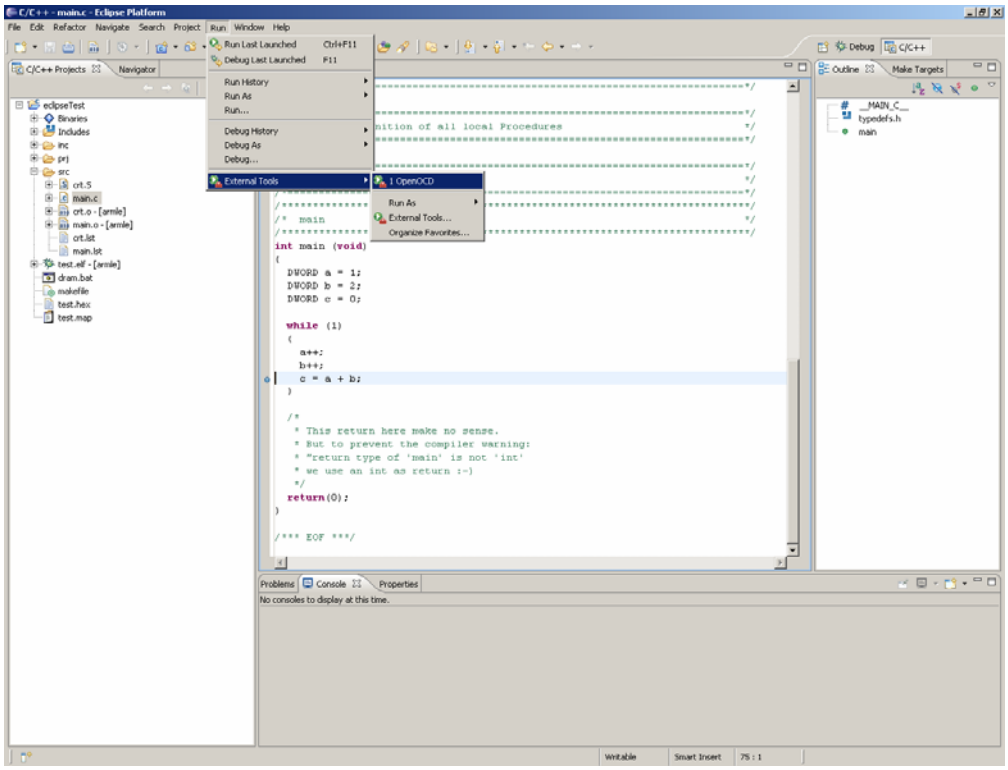


On the subsequent window, check the box for OpenOCD and click “OK”.

Now click "OK" on the "Organize External Tools…" and OpenOCD will now appear in the Run pull-down menu.

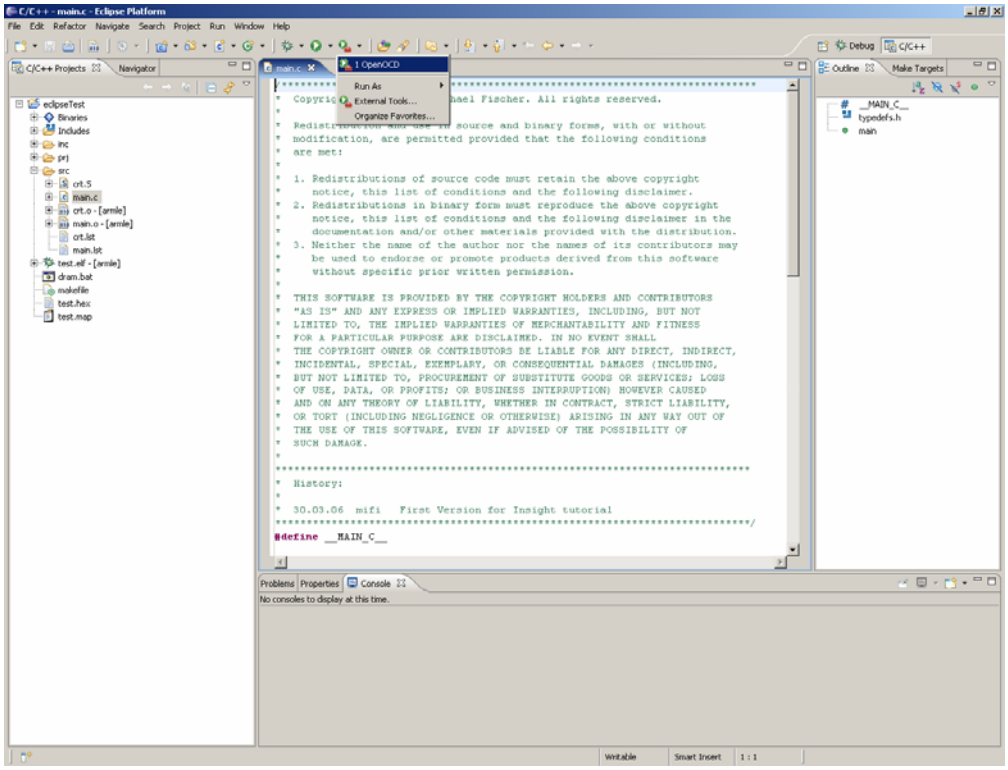


Now when we click on the Run pull-down menu and select "External Tools", we see that OpenOCD is present at the top of the pull-down list.

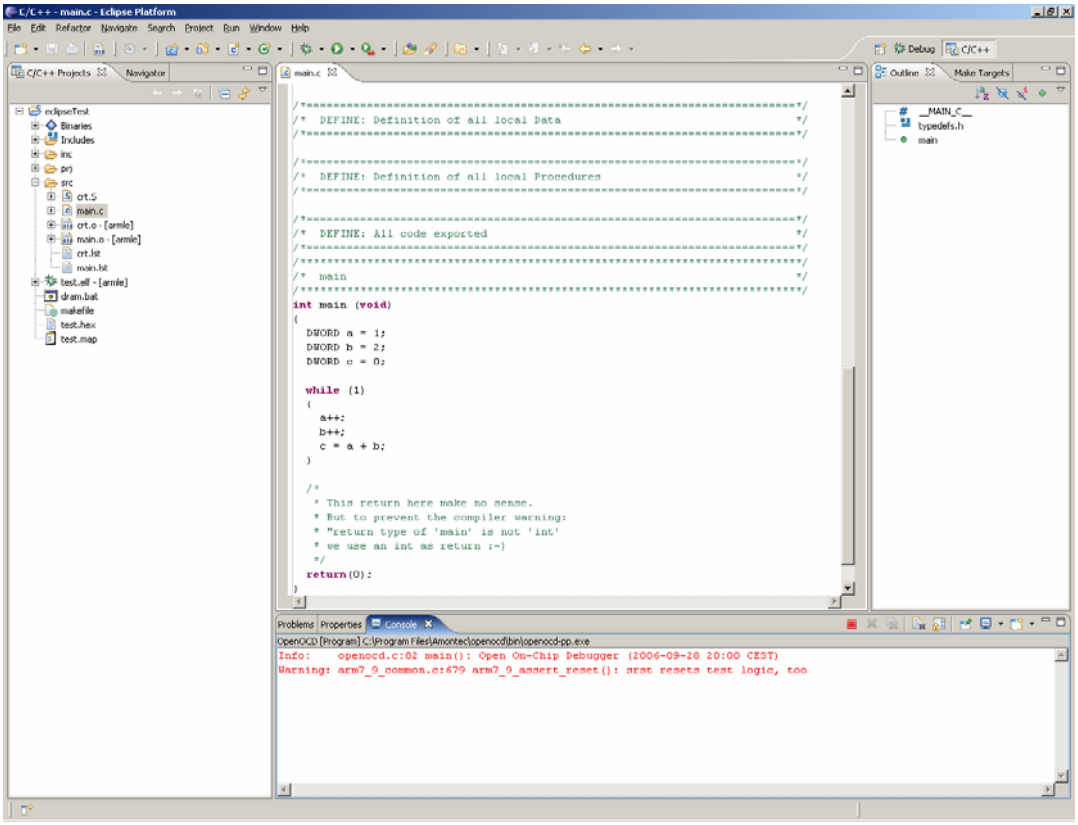


An even more convenient way to start OpenOCD is to use the “External Tolls” tool button in the toolbar.

Specifically click on the down arrow to reveal the external tools that have been set up. Eclipse remembers the last External Tool you selected, so clicking on just the red toolbox will start the previous tool.



By clicking on OpenOCD, the OpenOCD utility will start and show information in the console window, as shown below.



A.1 CDMS.h

```
/*-----------------------------------------------------------------------*/
/* File Name:           CDMS.h                                           */
/* Object:              CDMS Prototype Board Features Definition File.   */
/*                                                                       */
/*                      Creation from example eb55.h.                    */
/*-----------------------------------------------------------------------*/

#ifndef CDMS_h
#define CDMS_h

#include "AT91M55800A.h"
#define __inline inline

/*-----------------------------------*/
/* CDMS External Memories Definition */
/*-----------------------------------*/

/* Flash Memory NCS0: S29AL016D 1M*16 */
#define FLASH0_BASE     (0x01000000)
#define FLASH_SIZE      (2*1024*1024)               /* byte */

/* Flash Memory NCS1: S29AL016D 1M*16 */
#define FLASH1_BASE     (0x02000000)
#define FLASH_SIZE      (2*1024*1024)               /* byte */

/* Flash Memory NCS2: S29AL016D 1M*16 */
#define FLASH2_BASE     (0x03000000)
#define FLASH_SIZE      (2*1024*1024)               /* byte */

/* SDRAM Memory NCS3: R1LV0416C 256k*16 */
#define EXT_SRAM_BASE   (0x04000000)
#define EXT_SRAM_SIZE   (2*256*1024)                /* byte */
```

```
/*-------------------------------------------------------------------*/
/*                      EBIInitializationData                        */
/*-------------------------------------------------------------------*/
/*                      4 MHz master clock assumed.                  */
/*-------------------------------------------------------------------*/

/* 0x01000000, 16bits data bus, no wait states, 16MB pages, 2 TDF, only
16bit capable */
#define EBI_CSR_0       ((unsigned int )(FLASH0_BASE | 0x3501 ))

/* 0x02000000, 16bits data bus, no wait states, 16MB pages, 2 TDF, only
16bit capable */
#define EBI_CSR_1       ((unsigned int )(FLASH1_BASE | 0x3501))

/* 0x03000000, 16bits data bus, no wait states, 16MB pages, 2 TDF, only
16bit capable */
#define EBI_CSR_2       ((unsigned int )(FLASH2_BASE | 0x3501))

/* 0x04000000, 16bits data bus, no wait states, 16MB pages, 2 TDF, only
16bit capable */
#define EBI_CSR_3       ((unsigned int )(EXT_SRAM_BASE | 0x3501))

#define EBI_CSR_4       ((unsigned int )0x40000000)            /* unused */
#define EBI_CSR_5       ((unsigned int )0x50000000)            /* unused */
#define EBI_CSR_6       ((unsigned int )0x60000000)            /* unused */
#define EBI_CSR_7       ((unsigned int )0x70000000)            /* unused */

/*------------------------------*/
/* Software I2C bus Definition */
/*------------------------------*/

#define SCL             AT91C_PB20
#define SDA             AT91C_PB21
#define PIO_SCL         1
#define PIO_SDA         2

/*------------------------------*/
/*   Push Buttons Definition.  */
/*------------------------------*/
#define SW1_MASK        AT91C_PIO_PB4
```

```
/*-----------------------------*/
/*         A/D Converter       */
/*-----------------------------*/
#define AD_NCONVST      AT91C_PB19
#define START_CONV      0x0

#define LOOP_DAC1_AD0
#define LOOP_DAC0_AD4

/*-----------------------------*/
/*          Master Clock       */
/*-----------------------------*/

#define MCK             4000000
#define MCKKHz          (MCK/1000)

#define EXT_QUARTZ      20000000

#endif /* CDMS_h */
```

A.2 Main.c

```
/*-------------------------------------------------------------------------*/
/*                      main.c                                             */
/*               Main function for the test of the CDMS                    */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*                      Function prototypes                                */
/*-------------------------------------------------------------------------*/
void IRQ_Routine (void)   __attribute__ ((interrupt("IRQ")));
void FIQ_Routine (void)   __attribute__ ((interrupt("FIQ")));
void SWI_Routine (void)   __attribute__ ((interrupt("SWI")));
void UNDEF_Routine (void) __attribute__ ((interrupt("UNDEF")));

/*-------------------------------------------------------------------------*/
/*                           Header files                                  */
/*-------------------------------------------------------------------------*/
#include "AT91M55800A.h"
#include "CDMS.h"
```

```
/*---------------------------------------------------------------------------*/
/*                            Global Variables                               */
/*---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------*/
/*                            External References                            */
/*---------------------------------------------------------------------------*/
extern      void LowLevelInit(void);

/*---------------------------------------------------------------------------*/
/*                                  MAIN                                     */
/*---------------------------------------------------------------------------*/
int   main (void)
{
  // Initialize the Atmel AT91M55800A
  LowLevelInit();
  //Write your code here…
}

/*---------------------------------------------------------------------------*/
/*                      Stubs for various interrupts                         */
/*---------------------------------------------------------------------------*/

void IRQ_Routine (void) {    while (1);  }

void FIQ_Routine (void)  {   while (1);  }

void SWI_Routine (void)  {   while (1);  }

void UNDEF_Routine (void) {  while (1);  }
```

A.3 Makefile

```
NAME   = CDMSdemo

CC      = arm-elf-gcc
LD      = arm-elf-ld -v
AR      = arm-elf-ar
AS      = arm-elf-as
CP      = arm-elf-objcopy
OD    = arm-elf-objdump

CFLAGS  = -I./ -c -fno-common -O0 -g
AFLAGS  = -ahls -mapcs-32 -o crt.o
LFLAGS  =  -Map main.map -TCDMSdemo.cmd
CPFLAGS = --output-target=binary

ODFLAGS     = -x --syms

all: test

clean:
   -rm  crt.lst  main.lst  crt.o  main.o  lowlevelinit.o  main.elf  main.bin
main.hex main.map main.dmp

test: main.elf
   @ echo "...copying"
   $(CP) $(CPFLAGS) main.elf main.bin
   $(OD) $(ODFLAGS) main.elf > main.dmp

main.elf: crt.o main.o lowlevelinit.o CDMSdemo.cmd
   @ echo "..linking"
   $(LD) $(LFLAGS) -o main.elf  crt.o main.o lowlevelinit.o

crt.o: crt.s
   @ echo ".assembling"
   $(AS) $(AFLAGS) crt.s > crt.lst
main.o: main.c
   @ echo ".compiling"
   $(CC) $(CFLAGS) main.c
```

```
lowlevelinit.o: lowlevelinit.c
  @ echo ".compiling"
  $(CC) $(CFLAGS) lowlevelinit.c
```

A.4 Lowlevelinit.c

```
/*-------------------------------------------------------------------------*/
/* File Name:           lowlevelinit.c                                     */
/* Object:              CDMS Prototype Board low level initialization.     */
/* Adapted from EB55 startup initialization                                */
/*-------------------------------------------------------------------------*/
// Include the board file description
#include "AT91M55800A.h"
#include "CDMS.h"

extern void AT91F_Spurious_handler(void);
extern void AT91F_Default_IRQ_handler(void);
extern void AT91F_Default_FIQ_handler(void);

/*-------------------------------------------------------------------------*/
/* \fn    AT91F_LowLevelInit                                               */
/* \brief This function performs very low level HW initialization         */
/*        this function can be use a Stack, depending the compilation      */
/*        optimization mode                                                */
/*-------------------------------------------------------------------------*/
void LowLevelInit(void)
{
int            i;

 AT91PS_EBI     pEbi;
 AT91PS_AIC     pAic;
 AT91PS_APMC    pAPMC;

/*-------------------------------------------------------------------------*/
// Speed up the Boot sequence
// After reset, the number os wait states on chip select 0 is 8. All AT91
// Evaluation Boards fits fast flash memories, so that the number of wait
// states can be optimized to fast up the boot sequence.
/*-------------------------------------------------------------------------*/
```

```
// set sandart Wait State
        pEbi = AT91C_BASE_EBI ;
        pEbi->EBI_CSR[0] = EBI_CSR_0 ;


/*---------------------------------------------------------------------*/
//  Speed up the System Frequency
// After reset, the PLL must be setting the external Clock it fixed at 32 KHz
/*---------------------------------------------------------------------*/
        pAPMC = AT91C_BASE_APMC;
        pAPMC->APMC_CGMR  =  AT91C_APMC_MOSCEN  |  (AT91C_APMC_OSCOUNT  & (0x2F<< 16));


// Reading the APMC Status register to detect when the oscillator is stabilized
        while ( (pAPMC->APMC_SR & AT91C_APMC_MOSCS) != AT91C_APMC_MOSCS ) {} ;


// Commuting from Slow Clock to Main Oscillator (16Mhz)
        pAPMC->APMC_CGMR |= (AT91C_APMC_CSS & (0x1<< 14));


// Setup the PLL
        pAPMC->APMC_CGMR |= ( AT91C_APMC_MUL & (0x1 << 8)) | (3 << 24);


// Commuting from 4Mhz to PLL @ 20MHz
         pAPMC->APMC_CGMR   = (AT91C_APMC_CSS  &  (0x02  <<  14))  |  (pAPMC->APMC_CGMR & ~AT91C_APMC_CSS);


// Now the Master clock is the output of PLL @ 20MHz
/*---------------------------------------------------------------------*/
//  Set up EBI value
//--------------------
// After reset, All EBI register are setted at the default value
// The new value will be effective only after the remap command
/*---------------------------------------------------------------------*/
// Load System pEbi Base address and CSR0 Init Value
        pEbi->EBI_CSR[1] = EBI_CSR_1 ;
        pEbi->EBI_CSR[2] = EBI_CSR_2 ;
        pEbi->EBI_CSR[3] = EBI_CSR_3 ;
        pEbi->EBI_CSR[4] = EBI_CSR_4 ;
        pEbi->EBI_CSR[5] = EBI_CSR_5 ;
        pEbi->EBI_CSR[6] = EBI_CSR_6 ;
```

```
        pEbi->EBI_CSR[7] = EBI_CSR_7 ;
        // 6 memory regions, standard read
        pEbi->EBI_MCR = 6 ;
/*-------------------------------------------------------------------------*/
// Reset the Interrupt Controller
//-------------------------------
// Normally, the code is executed only if a reset has been actually performed.
// So, the AIC initialization resumes at setting up the default vectors.
/*-------------------------------------------------------------------------*/
// Load System pAic Base address
        pAic = AT91C_BASE_AIC;

// Mask All interrupt
        pAic->AIC_IDCR = 0xFFFFFFFF;

// Set up the default interrupt handler vectors
        pAic->AIC_SVR[0] = (int) AT91F_Default_FIQ_handler ;
        for (i=1;i < 31; i++)
        {
                pAic->AIC_SVR[i] = (int) AT91F_Default_IRQ_handler ;
        }
        pAic->AIC_SPU  = (int) AT91F_Spurious_handler ;
}
```